

Frontiers
in
Artificial
Intelligence
and
Applications

FINITE-STATE METHODS AND NATURAL LANGUAGE PROCESSING

Post-proceedings of the 7th International
Workshop FSMNLP 2008

Edited by
Jakub Piskorski
Bruce Watson
Anssi Yli-Jyrä

IOS
Press

VISIT...

LANZAROTE
Caliente.COM

FINITE-STATE METHODS AND NATURAL LANGUAGE PROCESSING

Frontiers in Artificial Intelligence and Applications

FAIA covers all aspects of theoretical and applied artificial intelligence research in the form of monographs, doctoral dissertations, textbooks, handbooks and proceedings volumes. The FAIA series contains several sub-series, including “Information Modelling and Knowledge Bases” and “Knowledge-Based Intelligent Engineering Systems”. It also includes the biennial ECAI, the European Conference on Artificial Intelligence, proceedings volumes, and other ECCAI – the European Coordinating Committee on Artificial Intelligence – sponsored publications. An editorial panel of internationally well-known scholars is appointed to provide a high quality selection.

Series Editors:

J. Breuker, R. Dieng-Kuntz, N. Guarino, J.N. Kok, J. Liu, R. López de Mántaras,
R. Mizoguchi, M. Musen, S.K. Pal and N. Zhong

Volume 191

Recently published in this series

- Vol. 190. Y. Kiyoki et al. (Eds.), Information Modelling and Knowledge Bases XX
- Vol. 189. E. Francesconi et al. (Eds.), Legal Knowledge and Information Systems – JURIX 2008: The Twenty-First Annual Conference
- Vol. 188. J. Breuker et al. (Eds.), Law, Ontologies and the Semantic Web – Channelling the Legal Information Flood
- Vol. 187. H.-M. Haav and A. Kalja (Eds.), Databases and Information Systems V – Selected Papers from the Eighth International Baltic Conference, DB&IS 2008
- Vol. 186. G. Lambert-Torres et al. (Eds.), Advances in Technological Applications of Logical and Intelligent Systems – Selected Papers from the Sixth Congress on Logic Applied to Technology
- Vol. 185. A. Biere et al. (Eds.), Handbook of Satisfiability
- Vol. 184. T. Alsinet, J. Puyol-Gruart and C. Torras (Eds.), Artificial Intelligence Research and Development – Proceedings of the 11th International Conference of the Catalan Association for Artificial Intelligence
- Vol. 183. C. Eschenbach and M. Grüninger (Eds.), Formal Ontology in Information Systems – Proceedings of the Fifth International Conference (FOIS 2008)
- Vol. 182. H. Fujita and I. Zulkernan (Eds.), New Trends in Software Methodologies, Tools and Techniques – Proceedings of the seventh SoMeT_08
- Vol. 181. A. Zgrzywa, K. Choroś and A. Siemiński (Eds.), New Trends in Multimedia and Network Information Systems

ISSN 0922-6389

Finite-State Methods and Natural Language Processing

Post-proceedings of the 7th International Workshop
FSMNLP 2008

Edited by

Jakub Piskorski

*Web Mining and Intelligence Group of IPSC, Joint Research
Centre of the European Commission, Ispra, Italy*

Bruce Watson

*FASTAR Research Group, Department of Computer Science,
University of Pretoria, Pretoria, South Africa*

and

Anssi Yli-Jyrä

*HFST Research Group, Department of General Linguistics,
University of Helsinki, Helsinki, Finland*

IOS
Press

Amsterdam • Berlin • Tokyo • Washington, DC

© 2009 The authors and IOS Press.

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, without prior written permission from the publisher.

ISBN 978-1-58603-975-2

Library of Congress Control Number: 2008943548

Publisher

IOS Press BV
Nieuwe Hemweg 6B
1013 BG Amsterdam
Netherlands
fax: +31 20 687 0019
e-mail: order@iospress.nl

Distributor in the UK and Ireland

Gazelle Books Services Ltd.
White Cross Mills
Hightown
Lancaster LA1 4XS
United Kingdom
fax: +44 1524 63232
e-mail: sales@gazellebooks.co.uk

Distributor in the USA and Canada

IOS Press, Inc.
4502 Rachael Manor Drive
Fairfax, VA 22032
USA
fax: +1 703 323 3668
e-mail: iosbooks@iospress.com

LEGAL NOTICE

The publisher is not responsible for the use which might be made of the following information.

PRINTED IN THE NETHERLANDS

Preface

These proceedings contain the final versions of the papers presented at the 7th International Workshop on Finite-State Methods and Natural Language Processing, FSMNLP 2008. The workshop was held in Ispra, Italy, on September 11–12, 2008. The event was the seventh instance in the series of FSMNLP workshops, and the third that was arranged as a stand-alone event. In 2008 FSMNLP was merged with the FASTAR workshop.

The aim of the FSMNLP workshops is to bring together members of the research and industrial community working on finite-state based models in language technology, computational linguistics, web mining, linguistics, and cognitive science on one hand, and, on related theory and methods in fields such as computer science and mathematics, on the other. Thus, the workshop series is a forum for researchers and practitioners working on applications as well as theoretical and implementation aspects. The special theme of FSMNLP 2008 centered around high performance finite-state devices in large-scale natural language text processing systems and applications.

In the context of FSMNLP 2008, we received in total 37 submissions, of which 13 were selected as regular papers, 6 as short papers and 1 as demo paper. The acceptance rate for regular papers was 46,4%. Most of the papers were evaluated by at least four Programme Committee members, with the help of external reviewers. Only 15% of the papers were reviewed by three reviewers. In addition to the submitted papers, four lectures were given by invited speakers. The invited speakers and the authors of the papers represented (at least) Croatia, Finland, France, Germany, Italy, Luxembourg, Netherlands, Portugal, Puerto Rico, Sweden, U.K., and the USA.

We would like to thank all workshop participants for their contributions and lively interaction during the two days. The presented papers covered a range of interesting NLP applications, including machine learning and translation, logic, computational phonology, morphology and semantics, data mining, information extraction and disambiguation, as well as programming, optimization and compression of finite-state networks. The applied methods included weighted algorithms, kernels, and tree automata. In addition, relevant aspects of software engineering, standardization, and European funding programmes were discussed.

We are greatly indebted to the members of the Programme Committee and the external referees for reviewing the papers and maintaining the high standard of the FSMNLP workshops. The members of the Programme Committee of FSMNLP 2008 were Cyril Allauzen (Google Research, New York, USA), Francisco Casacuberta (Instituto Tecnológico De Informática, Valencia, Spain), Jean-Marc Champarnaud (Université de Rouen, France), Maxime Crochemore (Department of Computer Science, King's College London, U.K.), Jan Daciuk (Gdańsk University of Technology, Poland), Karin Haenelt (Fraunhofer Gesellschaft and University of Heidelberg, Germany), Thomas Hanenforth (University of Potsdam, Germany), Colin de la Higuera (Jean Monnet University, Saint-Etienne, France), André Kempe (Yahoo Search Technologies, Paris, France), Derrick Kourie (Dept. of Computer Science, University of Pretoria, South Africa), Andras Kornai (Budapest Institute of Technology, Hungary and MetaCarta, Cambridge, USA), Marcus Kracht (University of California, Los Angeles, USA), Hans-Ulrich Krieger

(DFKI GmbH, Saarbrücken, Germany), Eric Laporte (Université de Marne-la-Vallée, France), Stoyan Mihov (Bulgarian Academy of Sciences, Sofia, Bulgaria), Herman Ney (RWTH Aachen University, Germany), Kemal Oflazer (Sabanci University, Turkey), Jakub Piskorski (Joint Research Center of the European Commission, Italy), Michael Riley (Google Research, New York, USA), Strahil Ristov (Ruder Boskovic Institute, Zagreb, Croatia), Wojciech Rytter (Warsaw University, Poland), Jacques Sakarovitch (Ecole nationale supérieure des Télécommunications, Paris, France), Max Silberztein (Université de Franche-Comté, France), Wojciech Skut (Google Research, Mountain View, USA), Bruce Watson (Dept. of Computer Science, University of Pretoria, South Africa) (PC co-chair), Shuly Wintner (University of Haifa, Israel), Atro Voutilainen (Connexor Oy, Finland), Anssi Yli-Jyrä (University of Helsinki and CSC – IT Center for Science, Espoo, Finland) (PC co-chair), Sheng Yu (University of Western Ontario, Canada), and Lynette van Zijl (Stellenbosch University, South Africa). The external reviewers were Marco Almeida, Marie-Pierre Beal, Oliver Bender, Jan Bungeroth, Pascal Caron, Loek Cleophas, Matthieu Constant, Stefan Hahn, Christopher Kermorvant, Sylvain Lombardy, Patrick Marty, Evgeny Matusov, Takuya Nakamura, Ernest Ketcha Ngassam, Jyrki Niemi, Sébastien Paumier, Maciej Pilichowski, Adam Przepiórkowski, Magnus Steinby, Yael Sygal, David Vilar, Hsu-Chun Yen, Francois Yvon, Artur Zaroda, and Djelloul Ziadi.

FSMNLP 2008 was organised by the Institute for the Protection and Security of the Citizen of the Joint Research Centre (JRC) of the European Commission in Ispra, Italy, in cooperation with the host of the next FSMNLP event, the FASTAR group of the University of Pretoria in South Africa. The Organizing Committee in 2008 had five JRC representatives: Regina Corradini, Daniela Negri, Jakub Piskorski (OC chair), Hristo Tanev, and Vanni Zavarella, and two members from the Department of Computer Science, University of Pretoria, South Africa: Derrick Kourie and Bruce Watson. A complementary role in long-term planning and coordination was played by the Steering Committee: Lauri Karttunen (Palo Alto Research Center, USA and Stanford University, USA) Kimmo Koskenniemi (University of Helsinki, Finland), Kemal Oflazer (Sabanci University, Turkey) and Anssi Yli-Jyrä (University of Helsinki and CSC – IT Centre for Science, Espoo).

The current year's event is pivotal to the series of FSMNLP workshops since it starts the tradition of organizing the workshops on a yearly basis. Locations for successive events, including FSMNLP 2008 in Ispra and FSMNLP 2009 in Pretoria were proposed already in FSMNLP 2007 in Potsdam. The success of FSMNLP 2008 indicates that there is a growing and wide interdisciplinary community with shared interest in finite-state methods and natural language processing. Therefore, we are looking forward to the FSMNLP 2009 that is to be held in Pretoria, South Africa next year!

In October 2008

*Jakub Piskorski
Bruce Watson
Anssi Yli-Jyrä*

Contents

Preface	v
<i>Jakub Piskorski, Bruce Watson and Anssi Yli-Jyrä</i>	
Invited Lectures	
CLARIN and Free Open Source Finite-State Tools	3
<i>Kimmo Koskenniemi and Anssi Yli-Jyrä</i>	
Learning with Weighted Transducers	14
<i>Corinna Cortes and Mehryar Mohri</i>	
Finite-State Machines for Mining Patterns in Very Large Text Repositories	23
<i>Wojciech Skut</i>	
Regular Papers	
The Kleene Language for Weighted Finite-State Programming	27
<i>Kenneth R. Beesley</i>	
Large-Scale Statistical Machine Translation with Weighted Finite State Transducers	39
<i>Graeme Blackwood, Adrià de Gispert, Jamie Brunning and William Byrne</i>	
Proper Noun Recognition and Classification Using Weighted Finite State Transducers	50
<i>Jörg Didakowski and Marko Drotschmann</i>	
Finite-State Local Grammars for Disambiguating Conjunctions in Portuguese Proper Names	62
<i>Samuel Eleutério and Elisabete Ranchhod</i>	
A Memory-Efficient ϵ -Removal Algorithm for Weighted Acyclic Finite-State Automata	72
<i>Thomas Hanneforth</i>	
Regular Expressions and Predicate Logic in Finite-State Language Processing	82
<i>Mans Hulden</i>	
Making Finite-State Methods Applicable to Languages Beyond Context-Freeness via Multi-Dimensional Trees	98
<i>Anna Kasprzik</i>	
Transducer Minimization and Information Compression for NooJ Dictionaries	110
<i>Slim Mesfar and Max Silberztein</i>	
Representing and Combining Calendar Information by Using Finite-State Transducers	122
<i>Jyrki Niemi and Kimmo Koskenniemi</i>	

Optimality Theory and Vector Semirings <i>Wolfgang Seeker and Daniel Quernheim</i>	134
A Compression Method for Natural Language Automata <i>Lamia Tounsi, Béatrice Bouchou and Denis Maurel</i>	146
Event Extraction for Italian Using a Cascade of Finite-State Grammars <i>Vanni Zavarella, Hristo Tanev and Jakub Piskorski</i>	158
Short Papers	
Finite State Models for the Generation of Large Corpora of Natural Language Texts <i>Domenico Cantone, Salvatore Cristofaro, Simone Faro and Emanuele Giaquinta</i>	175
CroMo – Morphological Analysis for Standard Croatian and Its Synchronic and Diachronic Dialects and Variants <i>Damir Čavar, Ivo-Pavao Jazbec and Tomislav Stojanov</i>	183
Forest FIRE and FIRE Wood: Tools for Tree Automata and Tree Algorithms <i>Loek Cleophas</i>	191
An XML Format Proposal for the Description of Weighted Automata, Transducers and Regular Expressions <i>Akim Demaille, Alexandre Duret-Lutz, Florian Lesaint, Sylvain Lombardy, Jacques Sakarovitch and Florent Terrones</i>	199
A Simple Formalism for Capturing Reduplication in Finite-State Morphology <i>Mans Hulden and Shannon T. Bischoff</i>	207
Applying Finite State Morphology to Conversion Between Roman and Perso-Arabic Writing Systems <i>Jalal Maleki, Maziar Yaesoubi and Lars Ahrenberg</i>	215
Morphisto – An Open Source Morphological Analyzer for German <i>Andrea Zielinski and Christian Simon</i>	224
Subject Index	233
Author Index	235

Invited Lectures

This page intentionally left blank

CLARIN and Free Open Source Finite-State Tools¹

Kimmo KOSKENNIEMI², Anssi YLI-JYRÄ

Department of General Linguistics, University of Helsinki, Finland

Abstract. A new emerging European research infrastructure called CLARIN and a related project called HFST are briefly described. HFST has built a programming interface on top of some existing open source finite-state packages such as SFST and OpenFST. In order to verify its utility, HFST has built open source tools on top of this HFST interface. These tools create lexical transducers, compile morphophonological two-level rules and combine them into a transducer lexicon. The tools have been tested against independently created with full-scale lexicons and rules for Northern Sámi and Lule Sámi languages which have more complicated lexical and morphophonological structure than most other European languages.

Keywords. Research infrastructures, Finite state transducers, Morphological analysis, Software architecture

Introduction

Finite-state methods provide means for uniform treatment of morphology and other lower level phenomena in different languages. Even if the languages differ, the corresponding finite-state transducers (FST) can be used by the very same programs. Furthermore, different lexicon and rule formalisms can be compiled into technically similar FSTs if the programs just obey the same format for representing the FSTs.

It would be desirable to broaden the use of finite-state methods because it would make it easier to cope with the multitude of languages in Europe in a single framework. Because there are altogether perhaps about 100 majority and significant minority languages, a solution to this problem is both urgent and valuable for CLARIN which is committed to cope with all these languages.

Commercial proprietary finite-state tools for natural language processing (NLP) have been created by Xerox [1] and AT&T [2] and they are widely used in the researcher community. However, they are difficult to use in production systems partly because the licensing conditions and restrictions of use, and partly because their source code is not accessible for making one's own modifications or improvements.

There have been quite a few freely available open source packages of the basic finite-state tools, but only a handful of them are fully tested, have a reasonable documentation

¹This paper is part of the activities of CLARIN infrastructure project
<http://www.clarin.eu>

²Corresponding Author: Department of General Linguistics, P.O. Box 9, FI-00014 University of Helsinki, Finland, email: kimmo.koskenniemi@helsinki.fi.

and tutorials and are maintained on a long term basis. Types of such packages can be exemplified by FSA Utilities [3], MONA [4], and Tiburon [5]. Few of these packages have such open source licenses that cover simultaneously the needs of the industry, the academy and common research infrastructures. At the same time as basic FST packages have been reimplemented repeatedly, there has been a clear lack of more specialized open source tools such as lexicon and rule compilers that would produce transducers needed in lexical analysis and generation.

A project called HFST (Helsinki Finite-State Technology) was started in late 2007 with the aim of producing a programming interface on top of some existing free open source finite-state packages. It is expected to:

- provide a better documented programming interface which makes the building of new finite-state NLP tools more efficient and robust,
- allow the developer of a compiler to create code which is independent of a particular finite-state package, possibly changing the underlying package if that makes the tool run faster,
- allow comparisons of the efficiency of algorithms in different packages and stimulate further improvement of the packages and their algorithms,
- allow free combining of various transducers as Unix pipelines (in the spirit of the AT&T finite-state tools), and
- include weighted FSTs as a part of the basic finite-state tools.

1. CLARIN

CLARIN stands for *Common Language Resources and Technologies Research Infrastructure* and it is one of the 34 infrastructure projects listed in the ESFRI roadmap.³ CLARIN is one of the few infrastructure projects for humanities and it has now entered its 3 year preparatory phase (EC Grant FP7-RI-2122230). CLARIN represents practically all EU member states and has currently some 120 member organizations. The CLARIN preparatory phase project has 32 partner organizations. CLARIN is coordinated by Steven Krauwer at the University of Utrecht.

At numerous centers and institutions around Europe, there are lots of language resources and tools of several types including (but not limited to):

- computer readable text corpora possibly with grammatical or other annotation,
- speech corpora with possible transliteration and annotations,
- multimedia recordings of conversations, child language learning, etc.,
- computer readable dictionaries and other lexical materials,
- tools, i.e. programs for parsing and processing of the above data,
- metadata and taxonomies describing the above types of materials,
- standards and norms governing the formats of various materials and the input and output of the tools.

³See <http://cordis.europa.eu/esfri/roadmap.htm> for more information on European Strategic Forum for Research Infrastructures (ESFRI) and <http://www.clarin.eu> for more details of CLARIN and its organization.

In addition to those specialized centers, large amounts of relevant language materials exist at digital libraries, stored by the publishers and printers, and in various archives for television and radio broadcastings. As digitizing becomes more practical, we can expect that almost all relevant language materials will be digitized sooner or later.

At first glance, even ordinary present day technology, i.e. the Internet, conventional servers, and desktop computers could handle these volumes of language data and provide the researchers an easy access to all these materials and tools. The resources and tools are, however, fragmented in several senses:

- It is difficult to find out whether certain types of materials or tools exist and if they exist, where they are.
- Even if the researcher would find the resources or tools needed, it is difficult to see how to get permission for using them. On the other hand, it is currently not even easy for the possessor of such materials to grant the necessary permissions. Most materials are constrained by copyright, and many recordings contain sensitive personal data restricted by ethical codes of conduct.
- Finally, even if one succeeds in getting the permissions, then he/she probably cannot use the materials efficiently because the tools and the materials cannot be combined as such. If parts of the material are in different formats, or their annotations are incompatible, they cannot be combined without conversions and additional programming. Furthermore, the use of the tools will be difficult or impossible, if the input and output formats of the tools are not the same as what the materials follow, or the licenses of different tools prevent combining them.

The goal of CLARIN is to overcome all these obstacles by creating (1) systems with metadata to ease the locating of the resources, (2) a harmonized system for licensing, online authorization and authentication for enabling easy application and granting of permissions, and finally (3) standards for text and speech corpora, lexicons and tools which guarantee compatibility and interoperability. Also web services will be used for distributed and standardized communication among materials and tools, possibly located at different sites.

A set of CLARIN centers or hubs will be established, possibly by upgrading some of the existing service centers to meet the CLARIN norms of advanced networking and identity federation techniques. These centers will be the backbone of the interoperability, in particular for the metadata, authentication and authorization and web-based services. Other centers will probably connect to these hubs, or possibly share their materials and tools with these hubs.

In essence, the current preparatory phase is not funded to create materials and tools, i.e. the contents of CLARIN. Instead, the preparatory phase sets up those standards, licensing patterns and formats needed for the interoperability. Nationally funded projects will make existing contents compatible with CLARIN services formats. They will also provide new CLARIN compatible materials and tools in the building phase of CLARIN. Ultimately CLARIN might contain all relevant European written and spoken materials, ancient and current, all smoothly accessible by the research community. The estimated cost of building CLARIN is more than 100 million euros. Already during the first year, the national CLARIN related funding has exceeded the funding the EC has granted to the preparatory phase of CLARIN (which was a total of 4.1 million euros for 3 years).

CLARIN is the infrastructure for the humanities. It will serve a wide spectrum of research which benefits from the seamless access to the European heritage of written docu-

ments and recorded speech as well as the study of specific languages. CLARIN will also encourage the creation of language technological tools and applications for individual languages according to the Basic Language Resource Kit (BLARK) [6] scheme.

The Work Package 5 of CLARIN concerns among other things the software tools to be included in CLARIN. A particular challenge here for the finite-state research is the fact that there are so many relevant European languages which ought to be handled on an equal basis. In order to enable searching and indexing, CLARIN has to define and provide a platform for processing all these languages. Language specific programs coded separately for each language are not practical for such an environment where so many languages are used in parallel. Robustness cannot be achieved, because if one of such programs contains a bug, the whole service might crash.

2. Benefits of Finite-State Techniques

Finite-state automata (FSA) and finite-state transducers (FST), both unweighted and weighted ones, are almost as simple devices there are which still can perform useful language and speech processing tasks [7,8]. They are mathematically fairly well understood and there are simple characteristics and measures which indicate their computational complexity, most notably the number of states of the machines.

Morphological analysis of a wide range of languages can be implemented using finite-state technologies in a straight-forward manner using pure finite-state transducers. This purity means essentially that the machines only have plain states and plain transitions and no ad hoc additions such as registers or special procedures or conditions embedded in states or transitions. Purity makes it possible to combine operations algebraically and optimize modules using formal theories. Pure transducers are bidirectional, i.e. in addition to analyzing forms, they can also be directly used for generating them.

Using pure finite-state machines also makes the common handling of many or all of the relevant languages attractive for the software engineer. Programs may easily process lots of different languages if the only thing a programmer needs to know of a language, is the name of the file containing a transducer for that language. One can, therefore use nontrivial rule based full scale inflectional dictionaries for any language in order to perform high precision and good recall searches, frequency calculations, lemmatization of online concordances etc. The software only needs to select an appropriate transducer in order to process each language correctly. The language specific transducers need to be created, of course, by computational linguists, but once they are available, the system can safely use any number of them.

The linguists and language technologist in the industry or within the academic research community can thus produce language modules as transducers. For morphological analysis, several tools and frameworks have been used. Well known tools include Xerox XFST⁴ with cascades of replace rules and LEXC transducer lexicons, morphological two-level analyzers such as the core of PC-KIMMO [9], morphological (unidirectional) analyzers such as HunMorph [10] and Malaga [11]. The remarkable thing is that these and many others can all be converted with reasonable effort into pure finite-state transducers which are technically similar to any FST.

⁴See <http://www.xrce.xerox.com/competencies/content-analysis/fssoft/home.en.html> for more information on the Xerox finite-state NLP tools.

The programmer of CLARIN services could do several sophisticated things for the 100 languages without language specific code, just by using different transducers according to the target language. One operation, the transduction can be used for a wide range of different tasks including:

- lemmatizing, i.e. reducing the inflected word-forms into their base form for indexing or information retrieval,
- disambiguation, of ambiguous word forms,
- generating search prefixes for retrieval of words in inflecting languages,
- generation of series of inflected forms to assist information retrieval,
- named entity recognition or other information extraction tasks,
- preprocessing of the input in order to standardize coding conventions or remove unnecessary annotations.

What we need are formalisms and compilers which convert various linguistic descriptions into finite-state transducers. A lexicon might need a different formalism than a morphophonological rule, and a named entity recognizer a different formalism than an information retrieval preprocessor. But descriptions of all these can be represented as transducers in a natural way.

Full scale syntactic description of languages is usually done with more powerful frameworks, even if finite-state devices are used in some parts of them. E.g. the local grammar framework in NooJ [12] (and INTEX) by Max Silberztein describes the phrase level information and larger syntactic constructions using finite-state networks. But these are not pure FSTs as they use some additional devices such as registers and tests in order to describe complicated linguistic phenomena. The maintainability and long-term support of special formalisms and impure network characteristics has to be addressed when we build common research infrastructures [13].

Impure finite-state systems can sometimes be converted mechanically into pure finite-state systems. In other cases, the testing and setting and reading of registers can perhaps be separated from the transduction and postponed to be performed after the application of the transducers. With a general formalism for such setting and testing of registers, some generality may still be achieved.

3. HFST Finite-State Initiative

We have started an initiative called HFST aiming at a more unified and productive implementations of finite-state tools for language processing. We have three full time programmers during 2008 for HFST. The work is supervised by PhD Krister Lindén and the authors of this paper participate the work as advisors while Erik Axelsson programs the HFST interface, Miikka Silfverberg programs the HFST-TWOLC rule compiler, Tommi Pirinen, MA, programs the HFST-LEXC lexicon compiler. In addition, there are local research activities that use HFST and contribute to it.

Several libraries and compilers for finite-state calculus have been built during the past decades.⁵ Some commercial ones by Xerox [1] and AT&T [2] are well tested and efficient, but they are difficult and probably expensive to get for purposes other than

⁵See <https://kitwiki.csc.fi/twiki/bin/view/KitWiki/FsmReg> for a listing of a few dozens of finite-state packages.

academic research and testing. In order to be freely available, generally applicable and extensible resource production methods e.g. for morphology, the libraries and compilers need to meet additional requirements [14]. The commercial packages cannot be used freely and, in particular, it would be difficult to modify their code or combine them with some other software. The openness and extent of CLARIN might make it even more difficult to acquire appropriate licenses.

There are also several open source implementations of the finite-state calculus, e.g. SFST (by Helmut Schmid) [15], Vaucanson (Jacques Sakarovitch and the Vaucanson group) [16] and OpenFST (by M. Riley, J. Schalkwyk, W. Skut, C. Allauzen and M. Mohri) [17]. These are fairly well-matured and tested, and therefore most valuable for the implementers of further finite-state tools for natural language processing.

There are, however, some practical problems in using any one of these calculus packages as the basis for building another rule or lexicon compiler. The first problem is to choose the right one. The choice is difficult because there are several apparently good ones to choose of. Most packages fail to complement each other with interchangeable and extensible modules [18]. Furthermore, most packages have only limited documentation of their interface, data types and hidden assumptions and it will be difficult for the programmer to utilize and extend the existing code. Once the choice is made, the programmer becomes more or less permanently tied to that particular package, because one's code is shaped according to idiosyncrasies of that package: names and parameters of the functions being the easiest to observe, whereas special concepts and underlying conventions are more difficult even to notice. The undocumented features and assumptions will silently be used here and there in the code. Thus the choice of the package becomes soon irreversible.

3.1. Goals of the HFST

In order to ease the problems mentioned above HFST defines and documents a generic interface for the finite-state calculus so that the building of finite-state based tools would be easier and safer. The interface was first implemented for a few most promising basic finite-state calculus packages, initially for SFST and OpenFST. Thus, authors of finite-state based tools need not commit themselves to the data structures or underlying assumptions a particular underlying implementation of the finite-state calculus. The definition of a HFST programming interface serves some long and medium term goals:

- To enable the comparison of the underlying finite-state packages and to set up common concepts and terms for expressing expectations and claims on the packages. Thus, different packages can coexist in one neutral framework and be used through a single documented interface. We hope that this will also stimulate the competition among alternative finite-state packages and research groups and will lead improvement of the performance. In the best case, a critical mass of research is achieved which speeds up the development of underlying algorithms.
- To make it easier to build compilers for rule and lexicon formalisms because of the explicit documentation of functions and abstract data types in the interface. We hope that various research groups would make use of the HFST when building their own tools. We also hope that this will stimulate the research and development of formalisms to express various NLP problems and open source compilers for them.

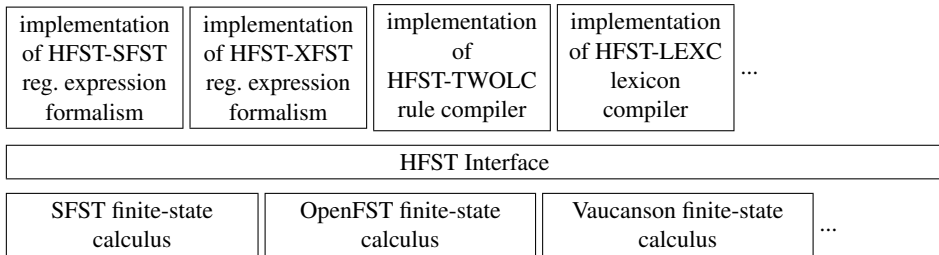


Figure 1. The Overview of the HFST Framework

- To stimulate the creation of morphological descriptions and corresponding transducer lexicons and other FST modules for the remaining European languages for which there are no adequate analyzers yet. We hope that this will be possible as more HFST based tools become available. This may be an attractive option for those who are already acquainted with similar tools and concepts.

It is worth noting that most of the ca. 100 languages mentioned above, still lack adequate tools. They are languages having relatively few speakers, so there is no hope of getting commercial implementations just by waiting. Instead, language technology for those languages must be funded by public authorities, as was the case with the Sámi languages. Free open source software suits such languages well. Progress towards this goal serves the CLARIN purposes in a natural way.

3.2. Design of the HFST

The HFST framework consists of three main layers, see also figure 1:

1. At the bottom, there are the individual finite-state calculus packages as alternative modules: one for the SFST, another for OpenFST, and later on, one for Vaucanson, etc. These modules provide the basic operations of finite-state automata and transducers, such as union, concatenation, intersection and composition.
2. The HFST API interface which uses the first level modules to implement a more abstract view to the FSMs and FSTs. The interface defines abstract data structures and concepts which are common to all underlying bottom level packages. HFST relates this abstract representation to the bottom level data structures and functions by some minimal program code. The API interface is carefully and explicitly defined so that the user need not know anything of the internal solutions of the first level.
3. The application level of lexicon or rule compilers which are implemented with the knowledge of the HFST API and using its documentation. This level includes a reimplementations of (a) the SFST regular expression language interpreter, (b) the HFST-LEXC lexicon compiler, and (c) the HFST-TWOLC rule compiler. It will later on include interpreters e.g. for the Xerox XFST regular expression formalism and other (newer) formalisms.

The HFST API layer itself will be divided into sublayers. The lower ones define just the implementation of a minimal finite-state calculus and the higher sublayers defining

some useful derived concepts and operations for rule compilation, sub-expressions and alphabet manipulation. The most essential sublayers will be defined first, while higher sublayers may be inserted and changed until they mature. For example, the support for particular kinds of weight types and the fancy interpretations of auxiliary symbols are not among the concerns of the bottom layer of HFST API.

3.3. *Implementation of the HFST*

The HFST API is documented using the DOXYGEN system which generates HTML documentation out of inline comments we have put in the actual C++ program of the API. The documentation is available on the web site,⁶ where one can also find the user manuals, technical details, file formats and development ideas concerning HFST. HFST has put considerable effort to use consistent names for functions, parameters and data types and to establish the underlying concepts which are then used for the accurate definition of the logical effects of individual functions.

The HFST API has been made into as free open source software as possible. The API itself is neutral and allows switching between alternative FST calculus packages according to the user's application is created. However, the underlying FST calculus packages may pose some conditions, depending on their respective licenses: SFST is under GNU GPL license and OpenFST is under Apache license. A tool under such licenses is quite useful in the sense that it can be used for the creation of both open source FSTs (which is likely in case of languages with small number of speakers) and proprietary FSTs (which is a typical case for commercial language technology providers). GNU GPL and Apache licenses specifically guarantee the use of the programs for any purpose, be it academic or commercial. E.g. for the minority languages, it is essential that the resulting transducers can be used in open source applications such as OpenOffice Writer, but also in connection with commercial software such as Microsoft Office Word.

3.4. *First Tools as the Proof of Feasibility*

The HFST group has implemented the first applications and modules on the top of the HFST API. The purpose of these is to be test cases and demonstrations of the maturity of the interface layer. The most notable applications that have been realized so far reimplement well-known and widely used formalisms for lexicons and rules. These formalisms were ideal for purposes of testing of the API, and the recent results [19,20,21] simplified the methods needed to implement them as modules on top of the HFST API. The created modules include the following:

1. A lexicon compiler HFST-LEXC, which compiles lexicons into a FST in the same manner as the Xerox LEXC. The HFST-LEXC can cope with full scale dictionaries, multi-character symbols and regular expressions.
2. A two-level compiler HFST-TWOLC, which compiles two-level morphophonological rules into a set of transducers. The compiler also handles multi-context rules and detects possible rule conflicts and resolves them as appropriate.

⁶see <http://www.ling.helsinki.fi/kieliteknologia/tutkimus/hfst/> for documentation, downloading and the online HFST API interface.

3. For combining the compiled lexicon and the rules, a special program for Karttunen's intersecting composition operation [22] has been implemented. This program constructs the composition of the lexicon with the set-theoretic intersection of all TWOLC or HFST-TWOLC rules. The construction combines intersection and composition into a single step, which avoids the need for intersecting the rules by themselves and the risk to produce an intermediate result that is excessively large.

These steps were completed in October 2008 and tested with two independently built descriptions, one for Northern Sámi and the other for Lule Sámi. The lexicons and rules for these were created by a Sámi language technology project.⁷ The aim of the Sámi project is to build NLP tools such as spelling checkers for the major Sámi languages spoken in Norway. The descriptions were made initially using Xerox LEXC and TWOLC tools. The testing included sets of word-lists analyzed on the two systems, and a set of test cases for rules, all provided by the Sámi team. Identical results were produced with the Xerox and HFST tools.

A medium term aim of these initial implementations and tests is to encourage further projects to convert their existing morphological and lexical descriptions into HFST-LEXC and HFST-TWOLC formats which may be a relatively easy task in some cases. Teams who are familiar with the LEXC and TWOLC concepts might use the HFST-LEXC and HFST-TWOLC tools for creating new morphological analyzers as the tools and their documentation is readily available for downloading. Previously, the LEXC and TWOLC formalisms lacked fully compatible free implementations and defining specifications. The new HFST-based tools are the first fully compatible and freely available implementations of the well-known TWOLC and LEXC formalisms. They also complement the prior documentation of these formalisms in various tricky situations.

Previously, some uniformity to the design of new FST calculi packages has been created by AT&T's FSM Library, but there have not been attempts to combine so different finite-state calculi as SFST, OpenFST etc. Therefore comparison of alternative FST calculi has been a tricky and time-consuming task. The HFST-LEXC and HFST-TWOLC formalisms were implemented purely with the HFST API interface and they can now be compiled either with the SFST or the OpenFST engine for finite-state calculus (the former being somewhat faster).

HFST-TWOLC implements rule compilation algorithms by applying *Generalized Restriction* (GR) by Anssi Yli-Jyrä [19,20,21]. Only a part of the possibilities provided by GR are actually in use in HFST-TWOLC, but this formalism is an important step towards more general formalisms similar to Xerox' XFST. In fact, the known compilation formulas based on GR [21] are more general than the ones used in XFST. As a consequence, all types of two-level rules, and parallel, directed and ranked rewriting and mark-up rules could be handled in a uniform way using the GR formulas. Even the resolution of right-arrow rule conflicts in TWOLC can be accounted for with the coherent intersection operation of the GR.

We hope that the first HFST-based tools stimulate the field by attracting more researchers to experiment and create new tools. Building tools on the top of HFST API is easier than using packages such as SFST and OpenFST directly.

⁷See <http://giellatekno.uit.no/english.html> for more information.

3.5. Summary and Future Perspectives of the HFST

In sum, the HFST API and the derived formalisms such as HFST-LEXC and HFST-TWOLC contribute towards the following objectives

- improved documentation, maintainability, testing and harmonization of programming interfaces
- modularization and improved code reuse
- comparison and cross-fertilization of FST calculi packages.

The obvious next step is to define and implement new rule formalisms (or programming languages) by using the HFST API. This step involves several domains:

1. Lexicon formalisms with both contextual selectors and non-concatenative derivation, as well as weights indicating the probabilities of inflectional forms, derivational constructions and word roots: Weights of forms might be very useful for morphological disambiguation and guessing of the base form.
2. Morphophonological rules which assign probabilities of alternations: These might be very useful for treating dialectal, historical or otherwise non-standard forms of language. Another possible application could be in comparative linguistics when searching for cognates.
3. Rewrite formalisms, including variant application modes for parallel and ranked replace and mark-up rules [21]: New rewriting formalisms could be designed also specifically for text normalization, information extraction, named entity recognition, various task in information retrieval etc.
4. New formalisms for surface oriented syntactic analysis, including tagging and chunking, weighted grammars for bracket-encoded dependency and constituent structures [21], and for computational semantics of e.g. calendar expressions [23].
5. Along with richer knowledge-driven approaches, new formalisms would enable domain-specific approaches to machine learning of structured data (e.g. through string kernels, stochastic models and memory-based methods) and computer aided production of annotation, lexicons and treebanks plus numberless dynamic applications.

The list could be continued, and we invite other researchers to start projects of their own and share the experiences and ideas and build a larger community around the goal of developing finite-state methods for NLP.

References

- [1] Kenneth Beesley and Lauri Karttunen. *Finite state morphology*. CSLI Studies in Computational Linguistics. CSLI Publications, Stanford, CA, USA, 2003.
- [2] Mehryar Mohri, Fernando C. N. Pereira, and Michael Riley. The design principles of a weighted finite-state transducer library. *Theoretical Computer Science*, 231(1):17–32, 2000.
- [3] Gertjan van Noord and Dale Gerdemann. An extendible regular expression compiler for finite-state approaches in natural language processing. In O. Boldt and H. Jürgensen, editors, *Automata Implementation. 4th International Workshop on Implementing Automata, WIA'99. Revised Papers.*, volume 2214 of *Lecture Notes in Computer Science*, Potsdam, Germany, July 17–19 2001.

- [4] Nils Klarlund and Anders Møller. *MONA Version 1.4 User Manual*. BRICS, Department of Computer Science, University of Aarhus, January 2001. Notes Series NS-01-1. Available from <http://www.brics.dk/mona/>. Revision of BRICS NS-98-3.
- [5] Jonathan May and Kevin Knight. Tiburon: A weighted tree automata toolkit. In Oscar H. Ibarra and Hsu-Chun Yen, editors, *CIAA*, volume 4094 of *Lecture Notes in Computer Science*, pages 102–113. Springer, 2006.
- [6] Steven Krauwer. The basic language resource kit (BLARK) as the first milestone for the language resources roadmap. Invited talk at SPECOM 2003, Moscow. Available from <http://www.elsnet.org/dox/krauwer-specom2003.pdf>, 2003.
- [7] Ronald M. Kaplan and Martin Kay. Regular models of phonological rule systems. *Computational Linguistics*, 20(3):331–378, 1994.
- [8] Mehryar Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–312, 1997.
- [9] Evan L. Antworth. *User's Guide to PC-KIMMO Version 2*. SIL International, 1995.
- [10] Viktor Trón, László Németh, Peter Halácsy, András Kornai, György Gyepesi, and Dániel Varga. Hunmorph: open source word analysis. In M. Jansche, editor, *Proceedings of ACL 2005 Software Workshop*, pages 77–85, 2005.
- [11] Björn Beutel. Malaga 7.12: User's and programmer's manual. Downloaded in October 2008 from <http://home.arcor.de/bjoern-beutel/malaga/>, 1995.
- [12] Max Silberstein. Nooj: a linguistic annotation system for corpus processing. In *Proceedings of HLT/EMNLP on Interactive Demonstrations*, pages 10–11, Morristown, NJ, USA, 2005. Association for Computational Linguistics.
- [13] Anssi Yli-Jyrä, Kimmo Koskenniemi, and Krister Lindén. Common infrastructure for finite-state methods and linguistics descriptions. In *International Workshop Towards a Research Infrastructure for Language Resources. LREC 2006 Workshop. May 22, 2006, Magazzini del Cotone Conference Center, Genoa, Italy*, 2006.
- [14] Anssi Yli-Jyrä. Toward a widely usable finite-state morphology workbench for less studied languages - part I: Desiderata. *Nordic Journal of African Studies*, 14(4 Special Issue):479–491, 2005.
- [15] Helmut Schmid. A programming language for finite state transducers. In Anssi Yli-Jyrä, Lauri Karttunen, and Juhani Karhumäki, editors, *FSMNLP*, volume 4002 of *Lecture Notes in Computer Science*, pages 308–309. Springer, 2005.
- [16] Sylvain Lombardy, Yann Régis-Gianas, and Jacques Sakarovitch. Introducing vaucanson. *Theoretical Computer Science*, 328:77–96, November 2004.
- [17] Cyril Allauzen, Michael Riley, Johan Schalkwyk, Wojciech Skut, and Mehryar Mohri. Openfst: A general and efficient weighted finite-state transducer library. In Jan Holub and Jan Zdárek, editors, *CIAA*, volume 4783 of *Lecture Notes in Computer Science*, pages 11–23. Springer, 2007.
- [18] Anssi Yli-Jyrä. Generality and openness in enabling methodologies for morphology and text processing. Genoa Cocosda & Write workshop presentations. COCOSDA 2006. 28th May 2006. Genoa, Italy, 2006.
- [19] Anssi Yli-Jyrä and Kimmo Koskenniemi. Compiling contextual restrictions on strings into finite-state automata. In *The Eindhoven FASTAR Days, Proceedings, 04/40, Computer Science Reports, Eindhoven, The Netherlands*, 2004.
- [20] Anssi Yli-Jyrä and Kimmo Koskenniemi. Compiling generalized two-level rules and grammars. In *Advances in Natural Language Processing, Proceedings of the 5th International Conference on NLP, FinTAL 2006, Turku, Finland, August 2006*, volume 4139 of *Lecture Notes in Artificial Intelligence*, pages 174–185. Springer, 2006.
- [21] Anssi Yli-Jyrä. Applications of diamonded double negation. In *Finite-State Methods and Natural Language Processing, 6th International Workshop, FSMNL-2007, Potsdam, Germany, September 14–16, Revised Papers*. Universitätsverlag, Potsdam, 2008.
- [22] Lauri Karttunen. Constructing lexical transducers. In *Proceedings of the 15th conference on Computational linguistics*, pages 406–411, Morristown, NJ, USA, 1994. Association for Computational Linguistics.
- [23] Jyrki Niemi and Lauri Carlson. Towards modeling the semantics of calendar expressions as extended regular expressions. *Proceedings of the 15th NODALIDA conference, Joensuu 2005, Ling@JoY, 1, 2006*, pages 133–138, 2005.

Learning with Weighted Transducers

Corinna CORTES^a and Mehryar MOHRI^{b,1}

^a *Google Research, 76 Ninth Avenue, New York, NY 10011*

^b *Courant Institute of Mathematical Sciences and Google Research,
 251 Mercer Street, New York, NY 10012*

Abstract. Weighted finite-state transducers have been used successfully in a variety of natural language processing applications, including speech recognition, speech synthesis, and machine translation. This paper shows how weighted transducers can be combined with existing learning algorithms to form powerful techniques for sequence learning problems.

Keywords. Learning, kernels, classification, regression, ranking, clustering, weighted automata, weighted transducers, rational powers series.

Introduction

Weighted transducer algorithms have been successfully used in a variety of applications in speech recognition [1,2,3], speech synthesis [4,5], optical character recognition [6], machine translation, a variety of other natural language processing tasks including parsing and language modeling, image processing [7], and computational biology [8,9]. This paper outlines the use of weighted transducers in *machine learning*.

A key relevance of weighted transducers to machine learning is their use in kernel methods applied to sequences. Weighted transducers provide a compact and simple representation of sequence kernels. Furthermore, standard weighted transducer algorithms such as composition and shortest-distance algorithms can be used to efficiently compute kernels based on weighted transducers.

1. Overview of Kernel Methods

Kernel methods are widely used in machine learning. They have been successfully used to deal with a variety of learning tasks including classification, regression, ranking, clustering, and dimensionality reduction. This section gives a brief overview of these methods.

Complex learning tasks are often tackled using a large number of features. Each point of the input space X is mapped to a high-dimensional feature space F via a non-linear mapping Φ . This may be to seek a linear separation in a higher-dimensional space,

¹Corresponding Author: Courant Institute of Mathematical Sciences and Google Research, 251 Mercer Street, New York, NY 10012; E-mail: mohri@cs.nyu.edu. Mehryar Mohri's work was partially funded by the New York State Office of Science Technology and Academic Research (NYSTAR).

which was not achievable in the original space, or to exploit other regression, ranking, clustering, or manifold properties easier to attain in that space. The dimension of the feature space F can be very large. In document classification, the features may be for example the set of all trigrams. Thus, even for a vocabulary of just 200,000 words, the dimension of F is 2×10^{15} .

The high dimensionality of F does not affect the generalization ability of large-margin algorithms such as support vector machines (SVMs). Remarkably, these algorithms benefit from theoretical guarantees for good generalization that depend only on the number of training points and the separation *margin*, and not on the dimensionality of the feature space. But the high dimensionality of F can directly impact the efficiency and even the practicality of such learning algorithms, as well as their use in prediction. This is because to determine their output hypothesis or to make predictions, these learning algorithms rely on the computation of a large number of dot products in the feature space F .

A solution to this problem is the so-called *kernel method*. This consists of defining a function $K: X \times X \rightarrow \mathbb{R}$ called a *kernel*, such that the value it associates to two examples x and y in input space, $K(x, y)$, coincides with the dot product of their images $\Phi(x)$ and $\Phi(y)$ in feature space:

$$\forall x, y \in X, \quad K(x, y) = \Phi(x) \cdot \Phi(y). \quad (1)$$

K is often viewed as a similarity measure. A crucial advantage of K is efficiency: there is no need anymore to define and explicitly compute $\Phi(x)$, $\Phi(y)$, and $\Phi(x) \cdot \Phi(y)$. Another benefit of K is flexibility: K can be arbitrarily chosen so long as the existence of Φ is guaranteed, which is called Mercer's condition. This condition is important to guarantee the convergence of training for algorithms such as SVMs. Some standard Mercer kernels over a vector space are the polynomial kernels of degree $d \in \mathbb{N}$, $K_d(x, y) = (x \cdot y + 1)^d$, and Gaussian kernels $K_\sigma(x, y) = \exp(-\|x - y\|^2 / \sigma^2)$, $\sigma \in \mathbb{R}_+$.

A condition equivalent to Mercer's condition is that the kernel K be *positive definite and symmetric* (PDS), that is, in the discrete case, the matrix $(K(x_i, x_j))_{1 \leq i, j \leq m}$ must be symmetric and positive semi-definite for any choice of n points x_1, \dots, x_m in X . Thus, the matrix must be symmetric and its eigenvalues non-negative.

The next section briefly describes a general family of kernels for sequences that is based on weighted transducers, *rational kernels*.

2. Rational Kernels

We start with some preliminary definitions of automata and transducers.

2.1. Weighted Transducers and Automata

Finite-state transducers are finite automata in which each transition is augmented with an output label in addition to the familiar input label [10,11]. Output labels are concatenated along a path to form an output sequence and similarly with input labels. *Weighted transducers* are finite-state transducers in which each transition carries some weight in addition to the input and output labels. The weights of the transducers considered in this paper are real values and they are multiplied along the paths. The weight of a pair of in-

put and output strings (x, y) is obtained by summing the weights of all the paths labeled with (x, y) . The following gives a formal definition of weighted transducers.

Definition 1. A weighted finite-state transducer T over $(\mathbb{R}, +, \cdot, 0, 1)$ is an 8-tuple $T = (\Sigma, \Delta, Q, I, F, E, \lambda, \rho)$ where Σ is the finite input alphabet of the transducer, Δ is the finite output alphabet, Q is a finite set of states, $I \subseteq Q$ the set of initial states, $F \subseteq Q$ the set of final states, $E \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times (\Delta \cup \{\epsilon\}) \times \mathbb{R} \times Q$ a finite set of transitions, $\lambda: I \rightarrow \mathbb{R}$ the initial weight function, and $\rho: F \rightarrow \mathbb{R}$ the final weight function mapping F to \mathbb{R} .

For a path π in a transducer, we denote by $p[\pi]$ the origin state of that path and by $n[\pi]$ its destination state. We also denote by $P(I, x, y, F)$ the set of paths from the initial states I to the final states F labeled with input string x and output string y . The weight of a path π is obtained by multiplying the weights of its constituent transitions and is denoted by $w[\pi]$. We shall say that a transducer T is *regulated* if the output weight associated by T to any pair of strings (x, y) by:

$$T(x, y) = \sum_{\pi \in P(I, x, y, F)} \lambda[p[\pi]] w[\pi] \rho[n[\pi]] \quad (2)$$

is in $\mathbb{R} \cup \{\infty\}$ and if this definition does not depend on the order of the terms in the sum. By convention, $T(x, y) = 0$ when $P(I, x, y, F) = \emptyset$. In the absence of ϵ -cycles, the set of accepting paths $P(I, x, y, F)$ is finite for any $(x, y) \in \Sigma^* \times \Delta^*$, and thus T is regulated. The transducers considered in this paper are all regulated. Figure 1 shows an example of a weighted transducer.

The standard rational operations, sum $+$, product or concatenation \cdot , and Kleene-closure $*$ can be defined for regulated transducers [12,13]. For any pair of strings (x, y) , and any three weighted regulated transducers T, T_1, T_2 ,

$$(T_1 + T_2)(x, y) = T_1(x, y) + T_2(x, y) \quad (3)$$

$$(T_1 \cdot T_2)(x, y) = \sum_{\substack{x_1 x_2 = x \\ y_1 y_2 = y}} T_1(x_1, y_1) T_2(x_2, y_2) \quad (4)$$

$$T^*(x, y) = \sum_{n=0}^{+\infty} T^n(x, y). \quad (5)$$

For any weighted transducer T , we denote by T^{-1} its *inverse*, that is the transducer obtained from T by swapping the input and output label of each transition. The *composition* of two weighted transducers T_1 and T_2 with matching input and output alphabets Σ , is a weighted transducer denoted by $T_1 \circ T_2$ when the sum:

$$(T_1 \circ T_2)(x, y) = \sum_{z \in \Sigma^*} T_1(x, z) T_2(z, y) \quad (6)$$

is well-defined and in \mathbb{R} for all $x, y \in \Sigma^*$ [12,13]. There exists an efficient algorithm for the composition of two weighted transducers [14,15]. The worst case complexity of that algorithm is quadratic, that is $O(|T_1||T_2|)$, where $|T_i|$ denotes the size of transducer T_i .

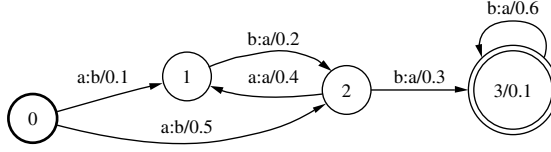


Figure 1. Example of a weighted transducer T . A bold circle indicates an initial state and a double-circle a final state. A final state carries a weight indicated after the slash symbol representing the state number. The initial weights are not indicated in all the examples in this paper since they are all equal to one. There are two paths in the transducer with input label abb and output label baa , thus the weight associated by T to the pair (abb, baa) is $T(abb, baa) = .1 \times .2 \times .3 \times .1 + .5 \times .3 \times .6 \times .1$.

2.2. Definition

As mentioned earlier, kernels can be viewed as similarity measures. It is often natural to define a similarity measure between two sequences, e.g., two documents or two biological sequences, as a function of the number of subsequences of some type that they share. These subsequences could be for example n -gram sequences, gappy n -grams, or substrings of any length. A sequence kernel is then typically defined as the sum of the product of the counts of these common subsequences.

Similarity measures of this kind can typically be computed using weighted finite-state transducers. This leads naturally to the following definition of a family of kernels over strings.

Definition 2. A kernel function $K : \Sigma^* \times \Sigma^* \rightarrow \mathbb{R}$ is rational when there exists a weighted transducer U such that $K(x, y) = U(x, y)$ for all sequences x and y .

Thus, for a rational kernel defined by U , $U(x, y)$ is the similarity measure between two strings x and y .²

2.3. Algorithm

$U(x, y)$ can be computed using the composition of weighted transducers [14,15]. Let M_x be a trivial weighted transducer representing x , that is a transducer such that $M_x(x, x) = 1$ and $M_x(y, z) = 0$ for $y \neq x$ or $z \neq x$. M_x can be constructed from a linear finite automaton representing x by augmenting each transition with an output label identical to the input label and by setting all transition and final weights to one. Similarly, we can construct a weighted transducer representing M_y . Then, by definition of composition, $(M_x \circ U \circ M_y)(x, y) = M_x(x, x)U(x, y)M_y(y, y) = U(x, y)$ and $(M_x \circ U \circ M_y)(z_1, z_2) = 0$ for $(z_1, z_2) \neq (x, y)$. Thus, $\sum_{u,v} (M_x \circ U \circ M_y)(u, v) = U(x, y)$, that is the sum of the weights of all paths of $M_x \circ U \circ M_y$ is exactly $U(x, y)$.

This gives a two-step algorithm to compute $K(x, y) = U(x, y)$: (a) use composition to compute $N = M_x \circ U \circ M_y$; (b) use a *shortest-distance algorithm* or forward-backward algorithm to compute the sum of the weights of all paths of N . We can assume that U does not contain any (ϵ, ϵ) cycle, that is a cycle with input ϵ and output ϵ . Otherwise, an equivalent weighted transducer without ϵ -transitions could be constructed from U by application of an ϵ -removal algorithm [17]. When U contains no ϵ -transition, N is necessarily acyclic since M_x and M_y are acyclic, and the computation of the sum of the

²This definition can be generalized to the case of an arbitrary semiring [16].

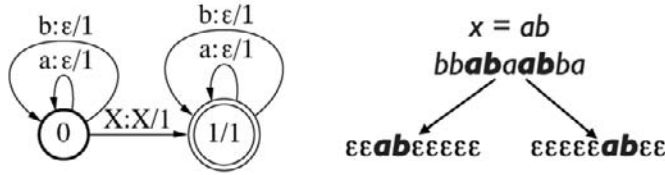


Figure 2. General count-based transducer T , for $\Sigma = \{a, b\}$. The figure illustrates the use of T in the special case where the automaton X accepts only the string $x = ab$, to count the number of occurrences of x in an input sequence such as $bbabaabba$.

weights of its paths can be done in linear time. Thus, the overall complexity of the computation of a rational kernel using that algorithm is $O(|U||M_x||M_y|)$, where $|U|$ remains constant in the calculation of a large number of kernels. In the particular case of many kernels used in practice, the complexity of the composition algorithm is in fact linear, which reduces the total cost of the application of the algorithm to $O(|U| + |M_x| + |M_y|)$. A new and more general n -way composition algorithm can also be used to dramatically improve the computational speed in other cases [18,19].

2.4. Properties

To guarantee the convergence of algorithms such as support vector machines, the rational kernel K used must be positive definite symmetric. The following theorem gives a general method for constructing a PDS rational kernel from any weighted transducer.

Theorem 1 ([16]). *Let T be an arbitrary weighted transducer, then $U = T \circ T^{-1}$ defines a PDS rational kernel.*

In this construction, the weighted transducer T can be viewed as the mapping from the input space $X = \Sigma^*$ to a high-dimensional feature space, compactly represented by the output of T . The construction of U from T is straightforward and very efficient since it requires only applying composition. Our inspection of the sequence kernels used in computational biology, natural language processing, or other sequence learning tasks, e.g., mismatch kernels [20], gappy n -gram kernels [21], locality-improved kernels [22], convolutions kernels for strings [23], tree kernels [24], n -gram kernels [16], and moment kernels [25], seem to show that they are all rational kernels of the form $T \circ T^{-1}$ [16]. In fact, we have conjectured that all PDS rational kernels are of this form and proven a number of results favoring that thesis [16].

Standard weighted transducer operations can be used to combine simpler PDS rational kernels to form more complex ones, as shown by the following theorem.

Theorem 2 ([16]). *PDS rational kernels are closed under sum, product, and closure operations.*

3. Applications

As already pointed out, to the best of our knowledge, the sequence kernels used in practice are all special instances of PDS rational kernels. Here we will briefly describe a

general and important family of rational kernels, *count-based kernels*, and show how a sequence kernel recently introduced in computational biology can be represented by as a weighted transducer.

3.1. Count-based kernels

The definition of many sequence kernels relies on the counts of some subsequences in the sequences x and y to compare. These subsequences can be of different nature, they may be for example arbitrary substrings, n -grams, gappy n -grams, or subsequences of ancestor sequences, where ancestor sequences are defined as sequences with a fixed number of mutations relative to the given sequence [20]. We will refer to such kernels as *count-based kernels*. These sequence kernels can typically be conveniently represented by weighted transducers and form rational kernels. This is because there exists a general weighted transducer that can be used to count the number of occurrences of the sequences described by an arbitrary regular expression.

Indeed, let X be an arbitrary finite automaton and thus representing an arbitrary regular expression. Then, the transducer defined by Figure 2 can be used to count all occurrences of the sequences accepted by X in any input sequence x . This is illustrated in the special case where X represents the single sequence $x = ab$ and for an input sequence $bbabaabba$.

The loop at the first state of T maps input symbols to ϵ until a match with a sequence in X is found. Then the sequence matched is mapped to itself and the remaining suffix of the input sequence mapped to ϵ at the final state of T . In the case of the sequence $bbabaabba$ and for the particular X considered, there are two possible occurrences of ab and thus two possible matches. The figure shows the alternative outputs generated by T . Since two paths are generated, each with weight one, the total weight associated by T to the input sequence is the sum two, which is the expected and correct count.

By theorem 1, transducer T can be used to construct a PDS rational kernel $U = T \circ T^{-1}$. This gives a very general method for the definition and construction of count-based sequence kernels. In fact, many sequence kernels successfully used in practice coincide precisely with this construction. This includes in particular n -gram kernels or gappy n -gram kernels.

3.2. Locality-improved kernel

A family of kernels was introduced by Zien et al. for the problem of *recognition of translation initiation sites* in computational biology. This problem consists of determining whether a start codon position in a DNA sequence is a translation initiation site (TIS).

The *locality-improved kernel* introduced by [22] is based on matching scores over windows of length $2l + 1$. It is defined as follows.

Definition 3 ([22]). *Let l and d be positive integers and $w_j, j \in [-l, +l]$ the weights associated to a match at position j in a window of size $2l + 1$. Then, the locality-improved kernel for two sequences x and y of length m is defined by*

$$K(x, y) = \sum_{p=1}^m \text{win}_p(x, y), \text{ with } \text{win}_p(x, y) = \left(\sum_{j=-l}^{+l} w_j \text{match}_{p+j}(x, y) \right)^d. \quad (7)$$

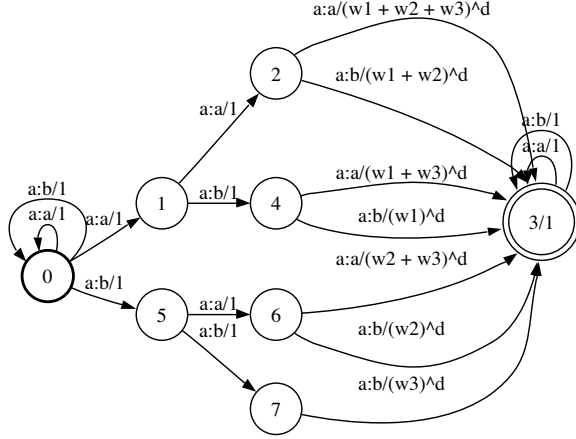


Figure 3. Fraction of the locality-improved kernel represented as a $U = T \circ T^{-1}$ rational kernel for the alphabet $\Sigma = \{a, b\}$ and $l = 1$. For the full transducer, all symmetric paths (obtained by exchanging a and b) should be added.

This kernel can be naturally combined with polynomial kernels to form more complex kernels and can be straightforwardly represented by a weighted transducer. Figure 3 shows the corresponding weighted transducer $U = T \circ T^{-1}$ for $\Sigma = \{a, b\}$ and $l = 1$ for the input sequence $x = aaa$. The corresponding weighted transducer T has the same topology as U , but the output label of all the transitions from state 0 to state 3 with a mismatch between input and output should instead be a special mismatch symbol, say z , and the transition weight should be the square root of the weight in U .

The loops of state 0 and 3 allow for arbitrary prefixes and suffixes around a window in which the mismatches are evaluated. U contains a unique path from state 0 to state 3 for each possible sequence of matches and mismatches. The weight of each sequence is marked at the last transition and equals the sum of the weights of the matching symbols taken to the power of d , the other transitions weights being one.

With this locality-improved kernel, Zien et al. obtain a 25% performance improvement over previous results on a task with about 13,500 sequences of which 3,300 are positive TIS examples and the rest are considered negative examples.

4. Conclusion

Weighted transducers give a general framework for the representation and computation of sequence kernels. All sequence kernels used in natural language processing, computational biology, and other sequence-related tasks are special instances of rational kernels. This has an important algorithmic advantage since a single general and efficient algorithm can be used to compute such kernels. State-of-the-art implementations of the general algorithms for the use of weighted transducers are available as part of the open-source software library OpenFst [26] and the algorithms for their use as sequence kernels exist as part of the open-source project OpenKernel library [27], freeing up the machine learning practitioner to focus on designing effective kernels for the problem at hand. The OpenKernel library interfaces with the popular software package LIBSVM [28] for easy experimentation with novel PDS rational kernels for classification and regression tasks.

Any weighted transducer T can be used to define a PDS sequence kernel by composing it with its inverse, and existing PDS rational kernels can be combined via standard rational operations to defined more complex PDS rational kernels. This has an important consequence for the design and improvement of sequence kernels. Furthermore, the graphical representation of rational kernels makes it convenient to augment or modify them. For all these reasons, we believe that rational kernels constitute just the *right* algorithmic and representational framework for sequence kernels. Furthermore, sample points can be used to *learn* rational kernels themselves [29]. This helps optimally selecting the specific rational kernel, or the proper transition weights, for the learning task considered.

References

- [1] Mehryar Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23:2, 1997.
- [2] Mehryar Mohri. Statistical natural language processing. In M. Lothaire, editor, *Applied Combinatorics on Words*. Cambridge University Press, 2005.
- [3] Mehryar Mohri, Fernando C. N. Pereira, and Michael Riley. Speech recognition with weighted finite-state transducers. In Larry Rabiner and Fred Juang, editors, *Handbook on speech processing and speech communication, Part E: Speech recognition*. Springer-Verlag, Heidelberg, Germany, 2008.
- [4] Richard Sproat. A finite-state architecture for tokenization and grapheme-to-phoneme conversion in multilingual text analysis. In *Proceedings of the ACL SIGDAT Workshop, Dublin, Ireland*. ACL, 1995.
- [5] Cyril Allauzen, Mehryar Mohri, and Michael Riley. Statistical modeling for unit selection in speech synthesis. In *42nd Meeting of the Association for Computational Linguistics (ACL 2004), Proceedings of the Conference*, Barcelona, Spain, July 2004.
- [6] Thomas M. Breuel. The OCRopus open source OCR system. In *Proceedings of IS&T/SPIE 20th Annual Symposium*, 2008.
- [7] Jürgen Albert and Jarkko Kari. Digital image compression. In Manfred Droste, Werner Kuich, and Heiko Vogler, editors, *Handbook of weighted automata*, EATCS Monographs on Theoretical Computer Science. Springer, 2009.
- [8] Richard Durbin, Sean R. Eddy, Anders Krogh, and Graeme J. Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, Cambridge UK, 1998.
- [9] Cyril Allauzen, Mehryar Mohri, and Ameet Talwalkar. Sequence kernels for predicting protein essentiality. In *Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML 2008)*, Helsinki, Finland, July 2008.
- [10] Jean Berstel. *Transductions and Context-Free Languages*. Teubner Studienbücher: Stuttgart, 1979.
- [11] Samuel Eilenberg. *Automata, Languages and Machines*, volume A. Academic Press, 1974.
- [12] Arto Salomaa and Matti Soittola. *Automata-Theoretic Aspects of Formal Power Series*. Springer-Verlag: New York, 1978.
- [13] Werner Kuich and Arto Salomaa. *Semirings, Automata, Languages*. Number 5 in EATCS Monographs on Theoretical Computer Science. Springer-Verlag, Berlin-New York, 1986.
- [14] Fernando Pereira and Michael Riley. *Finite State Language Processing*, chapter Speech Recognition by Composition of Weighted Finite Automata. The MIT Press, 1997.
- [15] Mehryar Mohri, Fernando C. N. Pereira, and Michael Riley. Weighted automata in text and speech processing. In *Proceedings of the 12th biennial European Conference on Artificial Intelligence (ECAI-96), Workshop on Extended finite state models of language*, Budapest, Hungary, 1996. John Wiley and Sons, Chichester.
- [16] Corinna Cortes, Patrick Haffner, and Mehryar Mohri. Rational Kernels: Theory and Algorithms. *Journal of Machine Learning Research*, 5:1035–1062, 2004.
- [17] Mehryar Mohri. Generic Epsilon-Removal and Input Epsilon-Normalization Algorithms for Weighted Transducers. *International Journal of Foundations of Computer Science*, 13(1):129–143, 2002.
- [18] Cyril Allauzen and Mehryar Mohri. N-way composition of weighted finite-state transducers. Technical Report TR2007-902, Courant Institute of Mathematical Sciences, New York University, August 2007.

- [19] Cyril Allauzen and Mehryar Mohri. 3-way composition of weighted finite-state transducers. In *Proceedings of the 13th International Conference on Implementation and Application of Automata (CIAA 2008)*, volume 5148 of *Lecture Notes in Computer Science*, pages 262–273, San Francisco, California, July 2008. Springer-Verlag, Heidelberg, Germany.
- [20] Christina Leslie, Eleazar Eskin, Jason Weston, and William S. Noble. Mismatch String Kernels for SVM Protein Classification. In *NIPS 2002*. MIT Press, 2003.
- [21] Huma Lodhi, John Shawe-Taylor, Nello Cristianini, and Chris Watkins. Text classification using string kernels. In *NIPS 2000*, pages 563–569. MIT Press, 2001.
- [22] A. Zien, G. Rätsch, S. Mika, B. Schölkopf, T. Lengauer, and KR. Müller. Engineering support vector machine kernels that recognize translation initiation sites. *Bioinformatics*, 9(16):799–807, 2000.
- [23] David Haussler. Convolution Kernels on Discrete Structures. Technical Report UCSC-CRL-99-10, University of California at Santa Cruz, 1999.
- [24] Michael Collins and Nigel Duffy. Convolution kernels for natural language. In *NIPS 14*, Cambridge, MA, 2002. MIT Press.
- [25] Corinna Cortes and Mehryar Mohri. Moment Kernels for Regular Distributions. *Machine Learning*, 60(1-3):117–134, September 2005.
- [26] Cyril Allauzen, Michael Riley, Johan Schalkwyk, Wojciech Skut, and Mehryar Mohri. OpenFst: a general and efficient weighted finite-state transducer library. In *Proceedings of the 12th International Conference on Implementation and Application of Automata (CIAA 2007)*, volume 4783 of *Lecture Notes in Computer Science*, pages 11–23, Prague, Czech Republic, July 2007. Springer-Verlag, Heidelberg, Germany.
- [27] Cyril Allauzen and Mehryar Mohri. OpenKernel Library, 2007. <http://www.openkernel.org>.
- [28] Chih-Chung Chang and Chih-Jen Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at <http://www.csie.ntu.edu.tw/~cjlin/libsvm>.
- [29] Corinna Cortes, Mehryar Mohri, and Afshin Rostamizadeh. Learning sequence kernels. In *Proceedings of IEEE International Workshop on Machine Learning for Signal Processing (MLSP 2008)*, (invited lecture), Cancún, Mexico, October 2008.

Finite-State Machines for Mining Patterns in Very Large Text Repositories

Wojciech SKUT

Google Inc., 1600 Amphitheatre Pkwy, Mountain View, CA, 94043, USA
wvskut@gmail.com

Abstract. The emergence of WWW search engines since the 1990s has changed the scale of many natural language processing applications. Text mining, information extraction and related tasks can now be applied to tens of billions of documents, which sets new efficiency standards for NLP algorithms. Finite-state machines are an obvious choice of a formal framework for such applications. However, the scale of the problem (size of the searchable corpus, number of patterns to be matched) often poses a problem even to well-established finite-state string matching techniques. In my presentation, I will focus on the experience gained in the implementation a finite-state matching library optimized for searching large amounts of complex patterns in a WWW-scale repository of documents. Both algorithmic and implementation-related aspects of the task will be discussed. The library is based on OpenFST.

Keywords. search engines, text mining, finite-state machines, string matching, complex patterns, OpenFST

The slides of this talk are available and linked to the FSMNLP 2008 program at <http://langtech.jrc.it/FSMNLP2008/m/program.html>.

This page intentionally left blank

Regular Papers

This page intentionally left blank

The Kleene Language for Weighted Finite-State Programming

Kenneth R. BEESLEY¹

Text Analytics, Business Objects, an SAP Company

Abstract. Kleene is a high-level programming language for building, manipulating and testing weighted finite-state acceptors and transducers. It allows users to define networks using regular expressions and right-linear phrase-structure grammars, and it provides variables, functions and familiar control structures. Pre-edited Kleene scripts can be run from the command line, and a graphical user interface is provided for interactive programming and testing. The Kleene parser is implemented in JavaCC/JJTree, and the interpreter calls functions in the OpenFst library via the Java Native Interface (JNI). The design, implementation, development status and future plans for Kleene are described.

Keywords. Kleene, finite state programming language, OpenFst library

Introduction

Kleene is a programming language in the tradition of the AT&T Lextools [1],² the SFST-PL language [2],³ and the Xerox/PARC finite-state toolkit [3];⁴ all of which provide higher-level programming formalisms built on top of low-level finite-state libraries. Kleene allows programmers to specify weighted finite-state networks, including acceptors that encode regular languages and two-level transducers that encode regular relations, using both regular expressions and right-linear phrase-structure grammars. The language provides variables and functions, plus familiar control structures such as `if-elsif-else` statements and `while` loops.

The Java-language Kleene parser, implemented with JavaCC and JJTree [4],⁵ is Unicode-capable and portable. Successfully parsed statements are reduced to abstract syntax trees (ASTs); and the Java-language interpreter, implemented using the visitor design pattern,⁶ interprets the ASTs by calling C++ functions in the OpenFst finite-state library [5]⁷ via the Java Native Interface (JNI) [6,7].

¹Business Objects, P.O. Box 540475, North Salt Lake, Utah 84054, USA; E-mail: ken.beesley@sap.com

²<http://www.research.att.com/~alb/lextools/>

³<http://www.ims.uni-stuttgart.de/projekte/gramotron/SOFTWARE/SFST.html>

⁴<http://www.fsmbok.com>

⁵<https://javacc.dev.java.net>

⁶http://en.wikipedia.org/wiki/Visitor_pattern

⁷A beta version of the OpenFst library (<http://www.openfst.org>) was first released 11 July 2007. The OpenFst library, and additional C++ code that wraps the OpenFst functions so that they can be called from Java, must be recompiled for each platform.

The Kleene GUI, implemented with the Java Swing library, allows interactive creation, testing and graphic display of networks. The main application window, a Swing `JDesktopPane`, encloses a Unicode-capable terminal window, into which Kleene statements can be typed, and a symbol-table window that displays an icon for each defined network. Right-clicking on an icon triggers a pop-up menu with commands to delete, determinize, draw, etc. the associated network.

Like the `OpenFst` library on which it currently depends, Kleene will be kept free, open-source and Apache-licensed.⁸ The paper continues with a description of the design criteria for the Kleene language, the syntax, the current state of development, and plans for the future.

1. Design Criteria

The following requirements and desiderata have guided the design and implementation of Kleene.

1. The Kleene language must be compelling, easy to learn and well documented. The syntax and semantics should always aim for maximum familiarity and “least astonishment”.
2. Programmers must be able to run pre-edited scripts and type statements interactively into a GUI.
3. Kleene will allow networks to be defined using both regular expressions and right-linear phrase-structure grammars.
4. The syntax should follow, as far as is possible and appropriate, the familiar syntax of Perl-like regular expressions. Non-regular features of Perl regular expressions, such as back-references, will be excluded; and operators will be added to denote weights, relations, subtraction, complementation and intersection, which are lacking in Perl-like regular expressions.
5. The abstraction mechanisms will include variables, built-in functions and user-defined functions.
6. The syntax will include rule-like abbreviations similar in function to the Xerox/PARC Replace Rules [8,9,10,11,12] that denote regular relations and compile into finite-state transducers. Such rules can be used to encode phonological and orthographical alternations in morphological analyzer/generators and phoneme-to-phone alternations in speech applications.
7. Unicode will be supported from the beginning, not only in data strings, but also in Kleene identifiers and operators.
8. The implementation should be maximally portable.
9. The implementation should be maximally modular, allowing the writing of interpreters based on various finite-state libraries that might become available.

⁸<http://www.apache.org/licenses/>

2. Syntax

2.1. Regular Expressions

In Kleene, regular expressions are the primary way to specify finite-state networks. The basic Kleene assignment statements have a regular expression on the right-hand side, e.g.

```
$myvar = (dog|cat|horse) s? ;
$yourvar = [A-Za-z] [A-Za-z0-9]* ;
$hisvar = ([A-Za-z]-[aeiouAEIOU]) + ;
$hervar = (bird|cow|elephant|pig) & (pig|ant|bird) ;
$ourvar = (dog):(chien) o (chien):(Hund) ;
$theirvar = [a-z]+ ( a <0.91629> | b <0.51083> ) ;
```

2.1.1. Primary Regular Expressions

Primary regular expressions are recognized directly by the tokenizer and so are effectively of highest precedence.

a b c	simple alphabetic symbols
.	(dot) matches any character
* \+ \? \.	literalized special characters
' [Noun] ' ' +Noun '	single symbols with multi-character names
\$myvar \$foo	names of variables denoting a network

2.1.2. Inherently Delimited Regular Expressions

The following regular expressions are syntactically complex but inherently delimited, making them also of highest precedence.

[aeiou] [A-Za-z0-9]	character sets (unions)
[^aeiou] [^A-Za-z0-9]	complemented character sets
"dog" "+" "AT&T"	double-quoted concatenations of symbols
<0.5> <0.01>	weights
\$&myfunction(args...)	call to a function returning a network
\$@mynetarray[n]	reference to an element of an array of networks

Kleene employs a system of sigils⁹ to distinguish identifiers like \$abc from simple concatenations of symbols like abc. A prefixed \$ marks a variable name with a finite-state network value; a prefixed \$& marks the name of a function that returns a network value; and a prefixed \$@ marks the name of an array of networks. Note the distinction between a single-quoted multi-character symbol like ' [Noun] ', which denotes a single symbol with a multi-character print name, versus a double-quoted string like "dog", which denotes the concatenation of the individual literal symbols between the quotes. Double quoting is not needed for normal alphabetic symbols – the regular expression "dog" is equivalent to dog, d o g, etc. – but is useful for literalizing special charac-

⁹[http://en.wikipedia.org/wiki/Sigil_\(computer_programming\)](http://en.wikipedia.org/wiki/Sigil_(computer_programming))

ters, e.g. "+", and for surrounding strings that include a special character, e.g. "AT&T" and "myfilename.txt".

2.1.3. Regular-Expression Operators

The following regular-expression operators are available, listed from high to low precedence.

()	parenthetical grouping	circumfix
:	crossproduct	infix
* + ? {2} {2,4} {2,} {,4}	iteration	postfix
~	language complement/negation	prefix
(no overt operator)	concatenation	juxtaposition
-	subtraction	infix
&	intersection	infix
	union	infix
(various rule operators)		
◦ or _o_	composition	infix

2.1.4. Precedence

The relative precedence of :, ~, and the various postfix iteration operators could still be debated, though there are precedents to follow.¹⁰ Currently, ~. * is equivalent to ~(. *) and so denotes the empty language; a : b * is equivalent to (a : b) *, and ~a : b is equivalent to ~(a : b).¹¹

By long tradition, concatenation has higher precedence than union and intersection, so one can write

```
$foo = dog|cat|mouse ;
```

to mean

```
$foo = (dog) | (cat) | (mouse) ;
```

Following other computer languages, & has slightly higher precedence than |.¹² The precedence of subtraction (-) relative to intersection (&) is still debatable, but it should probably be higher than union (|). Rules are typically composed together, so composition is given lower precedence than the various operators used to construct rules.¹³

Unicode is embraced from the beginning in the Java/Swing GUI, and users are encouraged, though not required, to edit their script files using a Unicode-capable text edi-

¹⁰The precedence shown is that currently favored by Helmut Schmid (SFST), and it's the relative precedence that Lauri Karttunen has chosen for the PARC *xfst* code to be released with the second printing of the book *Finite State Morphology* (private communications).

¹¹Transducers are not closed under complementation, so ~T, where T is a transducer, causes a runtime exception in Kleene, much like division by zero in arithmetic expressions.

¹²Similarly in most other languages, && has slightly higher precedence than ||.

¹³The alternation-rule syntax is not described in the present paper.

tor.¹⁴ Unicode characters can also be indicated in the syntax using the familiar `\uHHHH` and `\UHHHHHHHH` escape sequences, where each *H* is a hex digit 0-9, a-f or A-F.

2.1.5. Whitespace in Regular Expressions

Whitespace is ignored in regular expressions unless it is literalized. A space, for example, can be literalized in three ways:

1. Putting the literalizing backslash directly before the space, i.e. `_`
2. Putting the space inside `[...]` or `[^...]`, e.g. `[_abc]` matches a, b, c or a literal space, or
3. Putting the space inside double quotes, e.g. `"_"` and `"John_Smith"`

This is similar to the way that whitespace is ignored in arithmetic expressions, and it is like Perl regular expressions marked with the `/x` suffix.¹⁵

2.1.6. Denoting the Empty String

The empty (zero-length) string can be represented in various equivalent ways, including the Unicode U+03F5 GREEK LUNATE EPSILON SYMBOL ϵ ,¹⁶ the Unicode escape sequence `\u03F5`, the ASCII sequence `_e_`, an empty double-quoted string `"", a? - a`, etc.

2.2. Abstraction Mechanisms

2.2.1. Variables

As explained above, variables having a network value are distinguished syntactically with a `$` sigil, and they can appear on the left-hand side of an assignment statement.

`$foo = regularExpression ;`

The regular expression can continue over any number of lines, and the assignment statement is terminated with a semicolon. Once variables like `$foo` and `$bar` have been bound to network values, they can appear as operands in subsequent regular expressions.

```
$foo = dog | cat | elephant | zebra ;
$bar = bat | dog | octopus | frog ;
$result = ($foo | $bar) - (elephant | bat) ;
```

In this example, the resulting network would encode the language consisting of the strings “dog”, “cat”, “zebra”, “octopus” and “frog”. A reference to an unbound variable in a regular expression causes a runtime error.

¹⁴Kleene has a Java parser, and so it is able, using normal Java features, to read a file in almost any known encoding and convert it to Unicode. Unless told otherwise, Java assumes that a file being read is in the default encoding of the operating system and will convert it to Unicode accordingly.

¹⁵There are similar options in Python and Java to allow you to insert whitespace inside regular expressions to make them more readable.

¹⁶This Unicode character can be typed into the Kleene GUI, using standard Java Input Methods, including the CodePoint Input Method, and into any Kleene script prepared with a Unicode-capable text editor. The Unicode Standard specifies that the U+03F5 GREEK LUNATE EPSILON SYMBOL is for use in mathematical formulas, like regular expressions, and is not to be used in normal Greek text, where the U+03B5 GREEK SMALL LETTER EPSILON is appropriate.

Because finite-state networks can get very large, copying is avoided. The following sequence of instructions results in `$var2` being an alias for `$var1`, referencing the same network.

```
$var1 = a*b+[A-Za-z0-9]{3} ;
$var2 = $var1 ;
```

2.2.2. Built-in Functions

Rather than inventing and proliferating new regular-expression operators, the Kleene philosophy is to give access to some operations via built-in functions, including

```
$&invert(regex)
$&reverse(regex)
$&inputside(regex) or $&upperside(regex)
$&outputside(regex) or $&lowerside(regex)
$&copy(regex)
```

A function call that returns a network value is a Kleene regular expression and can, just like a variable having a network value, appear as an operand inside a larger regular expression.

2.2.3. User-defined Function Syntax

Users can also define and call their own functions. As a practical example, consider Priority Union, which is defined as follows:

Let Q and R be transducers. The priority union of Q and R , giving *input-side* priority to Q , returns the union of Q and R with the added restriction that if both Q and R share an input string i , then the output transducer contains only the paths from Q that have i on the input side.

Priority union with *output-side* priority is also potentially useful. In Kleene these functions can be defined as

```
&priority_union_input = &lambda($q, $r) {
    return $q | (~&inputside($q) _o_ $r) ;
}

&priority_union_output = &lambda($q, $r) {
    return $q | ($r _o_ ~&outputside($q)) ;
}
```

or using an alternative, and probably friendlier, syntax:

```
&priority_union_input($q, $r) = {
    return $q | (~&inputside($q) _o_ $r) ;
}

&priority_union_output($q, $r) = {
    return $q | ($r _o_ ~&outputside($q)) ;
}
```

Priority union can be useful in morphology to override regular but incorrect forms with their correct irregular forms. For example, assume that a network bound to `$productive_english` has been productively generated to contain input=>output string pairs like the following (where `[Verb]` and `[Past]` are multi-character symbols):

```
walk[Verb][Past] <=> walked
```

```
kick[Verb][Past] <=> kicked
```

```
think[Verb][Past] <=> thought
```

```
go[Verb][Past] <=> goed
```

Incorrect forms like **thought* and **goed* can be overridden by defining a smaller network encoding the correct mappings and priority-unioning it with `$productive_english`.

```
$corrections = (
    (dig):(dug)
|   (go):(went)
|   (say):(said)
|   (think):(thought)
) ('[Verb]' '[Past]'):"" ;

$english = $&priority_union_input($corrections,
                                $productive_english) ;
```

Once defined, functions can be called directly in regular expressions and used in the definition of yet other functions. For example, the normal composition of Q and R is $Q _o_ R$ (also typable as $Q \circ R$, using the Unicode RING OPERATOR, U+2218); and if the input-side language of Q is I , then the input-side language of $Q _o_ R$ may be a proper subset of I . That is, one or more of the original input strings of Q may not be accepted by the composition. The Lenient Composition of transducers Q and R accepts exactly the same input language as Q . The following definition of `$&lenient_composition_input()` is appropriate for the examples in Karttunen's regular formalization of Optimality Theory [13], where the `$base` transducer encodes a lexicon, and the `$filter` transducer encodes an optimality rule or filter being composed "underneath" the lexicon.

```
$&lenient_composition_input($base, $filter) = {
    return $&priority_union_input($base _o_ $filter, $base) ;
}
```

When, conversely, the rule or filter is being composed "on top of" the lexicon, and the desire is to preserve the output language of the lexicon, then the following `$&lenient_composition_output` is appropriate.

```
$&lenient_composition_output($filter, $base) = {
    return $&priority_union_output($filter _o_ $base, $base) ;
}
```

2.2.4. Function Call Semantics

Kleene maintains its environment as a directed graph of frames, where each frame contains a symbol table and both a dynamic link and a static link to other frames (or to null at the root of the environment). When a function is called, a new frame is allocated for its execution; the dynamic link of the new frame points back to the frame from which the function was called, and the static link points back to the frame where the function was defined.

The formal parameters of the function are bound, in the new frame’s local symbol table, to the passed-in argument values,¹⁷ and any variables introduced in the body of the function are also stored in the local symbol table. References to free (non-local) variables are resolved through the static link, thus implementing lexical scope. When the function terminates, it pushes the return value on the interpreter stack; then the calling frame, pointed to by the dynamic pointer, is once again made the current frame.

This fairly standard environment design supports functions that call other functions, functions that call themselves recursively, functions that themselves contain local definitions of functions, etc. Kleene also supports higher-order functions that return functions, as in the following example. Note that the `&&` sigil marks a function that returns a function that returns a network value.

```
&&append_suffix($suff) = {
    return &lambda($a) {
        return $a $suff ;
    }
}
&append_ing = &&append_suffix(ing) ;
&append_espVend = &&append_suffix(as|is|os|us|u|i) ;

$net1 = &append_ing(walk|talk) ;
$net2 = &append_espVend(pens|dir) ;
```

The function `&append_ing` just concatenates `ing` to its network argument, so `$net1` will be set to a network that encodes the language containing “walking” and “talking”. The function `&append_espVend` is designed to model the suffixation of Esperanto verb endings to verb roots, and `$net2` is set to a network that encodes the language containing “pensas”, “pensis”, “pensos”, “pensus”, “pensu”, “pensi”, “diras”, “diris”, etc.

2.3. Right-linear Phrase-structure Grammars

While regular expressions are formally capable of describing any regular language or regular relation, some linguistic phenomena – especially productive morphological compounding and derivation – can be awkward to model this way. Kleene therefore provides right-linear phrase-structure grammars that are similar in semantics, if not in syntax, to the Xerox/PARC `lexc` language [3]. While general phrase-structure grammars are context-free, requiring a push-down stack to parse, and so go beyond regular power, a right-linear (or left-linear) grammar is regular and so can be compiled into a finite-state network.

¹⁷Finite-state networks are passed by reference.

A Kleene phrase-structure GRAMMAR is defined as a set of PRODUCTIONS, each assigned to a variable with a $\$>$ sigil. Productions may include right-linear references to themselves or to other productions, which might not yet be defined. The productions are parsed immediately¹⁸ but are not evaluated until the entire grammar is built into a network via a call to the built-in function $\$&start(\$>StartProduction)$, which takes one production variable as its argument and treats it as the starting production of the whole grammar. The following example models a fragment of Esperanto noun morphotactics, including noun-root compounding.

```
 $\$>Root = ( kat \mid hund \mid elephant \mid dom ) ( \$>Root \mid \$>AugDim ) ;$ 
 $\$>AugDim = ( eg \mid et )? \$>Noun ;$ 
 $\$>Noun = o \$>Plur ;$ 
 $\$>Plur = j? \$>Case ;$ 
 $\$>Case = n? ;$ 

 $\$net = \$&start(\$>Root) ;$ 
```

The syntax on the right-hand side of productions is identical to regular-expression syntax, but allowing right-linear references to productions of the form $\$>Name$.

2.4. Arithmetic Expressions

While Kleene is designed primarily for creating and manipulating finite-state networks, it does support arithmetic expressions, variables that hold arithmetic values, functions that return arithmetic values, etc. Wherever a simple integer or float can appear in Kleene syntax, including numbered iterations like $a\{2\}$ and $b\{2,4\}$, and weights like $\langle 0.1 \rangle$, an arbitrarily complex arithmetic expression can appear, e.g. $a\{2+3\}$, $b\{2, \#maxlength - 1\}$ and $\langle \#defaultweight + .01 \rangle$.¹⁹

The arguments passed to a function could be any mix of regular expressions and arithmetic expressions, and one of the biggest challenges during Kleene development was the distinguishing and proper tokenization/parsing of the two separate expression types. For example, both use the plus sign as an operator, but in arithmetic expressions it is a binary infix operator of fairly low precedence, e.g. $2+3$, while in regular expressions it is a unary postfix operator of fairly high precedence, e.g. $ab+c$. Distinguishing the two expression types by inventing new operators – either for regular expressions or for arithmetic expressions – was judged to be completely unacceptable; and forcing users to surround regular expressions, or arithmetic expressions, with some kind of explicit delimiters, such as the Perl slashes $/.../$, was deemed inelegant and undesirable.

The solution adopted was to define a systematic set of sigils starting with $\$$ for network-value variables and functions, and $\#$ for arithmetic-value variables and functions. Parser lookahead distinguishes $\#a+\#b$ as an arithmetic expression, involving addition, from $\$a+\b , which is a regular expression indicating the concatenation of a Kleene-plussed network $\$a$ with network $\$b$. Once the sigil system is mastered, users can, in almost all cases, simply type familiar regular and arithmetic expressions in appropriate places.

¹⁸The parsed productions are stored as ASTs in the symbol table.

¹⁹Internally, Kleene stores arithmetic values as either a Long or a Double object.

The remaining problematic cases are expressions like 2 and 2+3, which start with digits. Are they arithmetic expressions, having an integer or float value, or regular expressions, having a network value? The Kleene solution is to treat bare digits by default as arithmetic expressions. To be interpreted as literal characters, and therefore regular expressions, digits must be literalized in the usual Kleene ways:

- Using the prefix backslash literalizer: \2
- Surrounding them with double quotes: "2", or
- Putting them in square-bracketed expressions: [2] [^2] [0-9] [^0-9]

2.5. Other Statements

In addition to network-, arithmetic- and function-assignment statements, Kleene provides *if-elsif-then* statements, *while* and *until* loops, iteration over array elements, and a variety of “housekeeping” statements for input-output, executing pre-edited scripts, retrieving information about networks, drawing networks, etc. For example, numbered iteration could be implemented by defining the following functions (though Kleene already provides the convenient {*n*} postfix operator):

```
$&iterate_by_recursion($net, #count) = {
    if (#count > 0) {
        return $net $&iterate_by_recursion($net, #count - 1) ;
    } else {
        return " " ;
    }
}

$&iterate_by_loop($net, #count) = {
    $result = " " ;
    while (#count > 0) {
        $result = $result $net ;
        #count = #count - 1 ;
    }
    return $result ;
}
```

3. Development Status

3.1. Current Status

At the time of writing, the following Kleene features are working:

- Interpretation of regular expressions and setting of variables
- Interpretation of arithmetic expressions and setting of variables
- Interpretation of right-linear phrase-structure grammars
- Definition and calling of functions that return network and arithmetic values
- Definition and calling of meta-functions that return functions
- Maintenance of identifier-value mappings in an environment of frames

- Automatic assignment of internal arc-label integers to syntactic symbols²⁰
- Implementation of a Unicode-friendly XML language for textual representation of networks²¹

Unicode is supported in Kleene to the extent that it is supported in Java and the Java Swing library – which is unusually well – and users can install and use their own Unicode fonts and Java Input Methods²² in their Java platforms. Some issues, including Unicode normalization and the expansion of the Kleene tokenizer to allow any Unicode letter character to appear in identifiers, still require some thought and work.

The Java wrapping of OpenFst functions and the GUI are well advanced, but not complete. Kleene currently runs only on Apple OS X, but because Java is unusually portable, and because OpenFst is also known to compile on Linux, a Linux port should not be difficult.

All of the examples shown in this paper are working.

3.2. *To Be Done*

Major planned features remaining to be implemented include:

- Interpretation of alternation rules
- Implementation of arrays, iteration over array elements, functions that return arrays, etc.
- Generalization of the interpreter, which currently handles only the default Tropical Semiring, to handle multiple semirings
- Writing practical runtime code and APIs to allow Kleene-built FSTs to be integrated easily into applications

Eventually, it is hoped to support graphical drag-and-drop programming of networks within the GUI.

4. Conclusion

The Kleene language provides a high-level programming language for creating, manipulating and testing weighted finite-state transducers. The current alpha implementation is still under active development, and anything could change. A beta release is planned for late 2008.

Like the OpenFst library on which it depends, Kleene will remain free, open-source and Apache-licensed to encourage collaboration and wide usage.

²⁰OpenFst automata store all arc labels as 32-bit integers. In Kleene, single symbols are automatically stored using their official Unicode code point values, including supplementary characters; and multi-character symbols are assigned an arbitrary code point value from a Unicode Private Use Area, starting with Plane 15. If necessary, for automata using unusually large alphabets of multi-character symbols, code values beyond the Unicode 21-bit range will be used, providing over 4 billion symbol distinctions.

²¹OpenFst still uses an ASCII-limited textual format inherited from AT&T (<http://www.research.att.com/~fsmtools/fsm/man4/fsm.5.html>).

²²<http://javadesktop.org/articles/InputMethod/index.html>

5. Acknowledgements

I am indebted to a number of people: first, to the whole OpenFst team, and especially Cyril Allauzen, for making the OpenFst library available and for providing patient explanations and advice; second, to my bosses Michael Wiesner and George Chitouras at Business Objects, an SAP Company, who appreciated the value of contributing to an open-source project; and third, to all those who have gone out of their way to provide information, help or encouragement, including Lauri Karttunen, André Kempe, Mike Wilkens, Helmut Schmid, Mike Maxwell, Natasha Lloyd, Anssi Yli-Jyrä and Kemal Oflazer. I have not always followed their advice, and any infelicities in the Kleene language and GUI are my responsibility.

References

- [1] Brian Roark and Richard Sproat. *Computational Approaches to Morphology and Syntax*. Oxford Surveys in Syntax & Morphology. Oxford University Press, Oxford, 2007.
- [2] Helmut Schmid. A programming language for finite state transducers. In *FSMNLP'05*, Helsinki, 2005.
- [3] Kenneth R. Beesley and Lauri Karttunen. *Finite State Morphology*. CSLI Publications, Palo Alto, CA, 2003.
- [4] Tom Copeland. *Generating Parsers with JavaCC*. Centennial Books, Alexandria, VA, 2007.
- [5] Cyril Allauzen, Michael Riley, Johan Schalkwyk, Wojciech Skut, and Mehryar Mohri. OpenFst: A general and efficient weighted finite-state transducer library. In *Proceedings of the Ninth International Conference on Implementation and Application of Automata (CIAA 2007)*, volume 4783 of *Lecture Notes in Computer Science*, pages 11–23. Springer, 2007.
- [6] Rob Gordon. *Essential JNI: Java Native Interface*. Prentice Hall PTR, Upper Saddle River, NJ, 1998.
- [7] Sheng Liang. *The Java Native Interface: Programmer's Guide and Specification*. Addison-Wesley, Reading, MA, 1999.
- [8] Lauri Karttunen. The replace operator. In *ACL'95*, Cambridge, MA, 1995. cmp-1g/9504032.
- [9] Lauri Karttunen and André Kempe. The parallel replacement operation in finite-state calculus. Technical Report MLTT-021, Xerox Research Centre Europe, Grenoble, France, December 1995. <http://www.xrce.xerox.com/publis/mltt/mltttech.html>.
- [10] Lauri Karttunen. Directed replacement. In *ACL'96*, Santa Cruz, CA, 1996. cmp-1g/9606029.
- [11] André Kempe and Lauri Karttunen. Parallel replacement in finite-state calculus. In *COLING'96*, Copenhagen, August 5–9 1996. cmp-1g/9607007.
- [12] Mehryar Mohri and Richard Sproat. An efficient compiler for weighted rewrite rules. In *ACL'96*, Santa Cruz, CA, 1996.
- [13] Lauri Karttunen. The proper treatment of optimality in computational phonology. In *FSMNLP'98. International Workshop on Finite-State Methods in Natural Language Processing*, Bilkent University, Ankara, Turkey, June 29 1998. cmp-1g/9804002.

Large-Scale Statistical Machine Translation with Weighted Finite State Transducers

Graeme BLACKWOOD, Adrià DE GISPert, Jamie BRUNNING and
William BYRNE

Machine Intelligence Laboratory
Department of Engineering, Cambridge University
Trumpington Street, Cambridge, CB2 1PZ, U.K.
 {gwb24, ad465, jjjb2, wjb31}@cam.ac.uk

Abstract. The Cambridge University Engineering Department phrase-based statistical machine translation system follows a generative model of translation and is implemented by the composition of component models of translation and movement realised as Weighted Finite State Transducers. Our flexible architecture requires no special purpose decoder and readily handles the large-scale natural language processing demands of state-of-the-art machine translation systems. In this paper we describe the CUED system's participation in the NIST 2008 Arabic-English machine translation evaluation task.

Keywords. Statistical machine translation, weighted finite state transducers, large-scale natural language processing, finite state grammars

Introduction

In the source-channel model of statistical machine translation [1], target sentences are viewed as source sentences that have passed through a noisy communication channel corrupting their surface form. The task of translation is to recover the source sentence that generated the observed target. The search for the best source sentence $\mathbf{S} = s_1, s_2, \dots, s_I$ for a given target $\mathbf{T} = t_1, t_2, \dots, t_J$ is typically inverted and decomposed as

$$\hat{\mathbf{S}} = \operatorname{argmax}_{\mathbf{S}} P(\mathbf{S}|\mathbf{T}) = \operatorname{argmax}_{\mathbf{S}} P(\mathbf{T}|\mathbf{S})P(\mathbf{S}), \quad (1)$$

where $P(\mathbf{T}|\mathbf{S})$ is the translation probability, $P(\mathbf{S})$ is the language model probability, and the argmax denotes the search for the best translation \mathbf{S} .

The Cambridge University Engineering Department statistical machine translation system follows the Transducer Translation Model (TTM) [2,3], a phrase-based generative model of translation that applies a series of transformations specified by conditional probability distributions and encoded as Weighted Finite State Transducers [4]. The main advantages are modularity, which facilitates the development and evaluation of individual components, and implementation simplicity, which allows us to focus on modelling

issues rather than complex decoding and search algorithms. The TTM scales naturally to very large data sets and no special-purpose decoder is required; by this we mean that standard WFST operations such as weighted composition can be used to obtain the 1-best translation or a lattice of alternative hypotheses. Finally, our system architecture readily supports speech translation, in which input ASR lattices can be translated in the same way as text [5].

1. The Transducer Translation Model

Under the Transducer Translation Model, the generation of target language sentence $\mathbf{T} = t_1^J$ starts with the generation of a source language sentence $\mathbf{S} = s_1^I$ by the source language model $P_G(s_1^I)$. Next, the source language sentence is segmented into phrases according to the unweighted uniform source phrasal segmentation model $P_W(u_1^K, K | s_1^I)$. This source phrase sequence generates a reordered target language phrase sequence according to the phrase translation and reordering model $P_R(x_1^K | u_1^K)$. Next, target language phrases are inserted into this sequence according to the insertion model $P_\Phi(v_1^R | x_1^K, u_1^K)$. Finally, the sequence of reordered and inserted target language phrases are transformed to word sequences t_1^J under the unweighted target phrasal segmentation model $P_\Omega(t_1^J | v_1^R)$. These component distributions together form a joint distribution over the source and target language sentences and their possible intermediate phrase sequences as $P(t_1^J, v_1^R, x_1^K, u_1^K, s_1^I)$.

In translation under the generative model, we start with the target sentence \mathbf{T} in the foreign language and then search for the best source sentence $\hat{\mathbf{S}}$. Encoding each distribution as a WFST leads to a model of translation as the series of compositions

$$L = G \circ W \circ R \circ \Phi \circ \Omega \circ T, \quad (2)$$

in which T is an acceptor for the target language word sequence and L is the word lattice of source language translations obtained during decoding. There is a direct correspondence between each distribution and the transducer in which it is realized (denoted by the distribution subscripts). The most likely translation $\hat{\mathbf{S}}$ is then the path in L with least cost (i.e. the minimum negative log-likelihood in the tropical semiring).

1.1. Phrase Reordering Transducers

The TTM reordering model is implemented by means of a phrase jump transducer, typically combined through composition with the one-state phrase translation WFST. In qualitative terms, this reordering model describes a jump sequence associated with each admissible permutation of the phrases [2]. In practice, it takes input source phrase sequences and outputs their translations in both monotonic and non-monotonic order.

In the simplest reordering model, known as MJ1-Flat, two adjacent phrases are allowed to swap positions with a fixed jump probability β_1 that is determined empirically. Figure 1 shows the WFST reordering transducer for the two phrases x_1 and x_2 . This simple model is effective since it significantly broadens the search space and, as source phrases can be arbitrarily long, individual words may move quite far in translation. However, it makes no distinction as to which phrases are more likely to be reordered in translation. This problem can be addressed by defining a separate jump probability $\beta_1(v_k, u_k)$

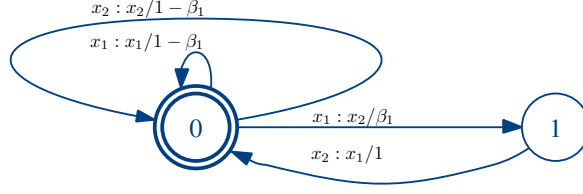


Figure 1. The MJ1-Flat reordering transducer for a sequence of two phrases “ $x_1 x_2$ ” with a fixed jump probability of β_1 .

for each phrase pair. The probabilities can be estimated from word alignments by examining adjacent phrase pairs and their orientation with respect to (v_k, u_k) and computing relative frequency estimates, in a similar fashion to Tillmann [6]. The actual WFST implementation is analogous to MJ1-Flat, but a new state is required for each phrase bigram, since the jump probability differs in each case.

1.2. Phrase Segmentation Transducers

In first-pass TTM translation all phrasal segmentations of a sentence are considered equally likely. The segmentation transducers are therefore unweighted and simply provide a mapping between the words and phrases of source and target language sentences. On the source side, the source language segmentation transducer W maps a source language word string to a lattice of all possible phrasal segmentations using the phrases of the phrase pair inventory. For example, if an acceptor for the source string “*exhibition of students returning from abroad*” is composed with the source language segmentation transducer, the result is the lattice of phrases shown in Figure 2. A similar segmentation process is applied to the target language sentence using the target language segmentation transducer Ω . The resulting lattice of phrases is the input to the decoding process in the TTM. Our flexible model architecture is such that additional inputs can be easily incorporated. For example, it may be useful to include alternative Arabic morphological analyses, variant Chinese character segmentations, or a lattice of recognition hypotheses output by an ASR system.

1.3. Language Model Acceptor

The language model $P_G(s_1^I)$ is encoded as weighted finite state acceptor G . The topology of this acceptor is such that states encode histories and arcs specify the n -gram con-

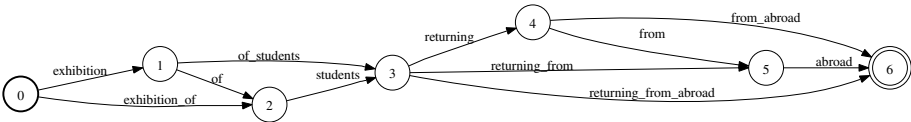


Figure 2. Phrase lattice encoding all possible segmentations of the source language string “*exhibition of students returning from abroad*” using only the phrases of the phrase pair inventory. The phrase label on each arc shows the constituent words of the phrase.

ditional probability of the labelled word given the history, or the context-specific backoff weight when there is no matching word. In first-pass translation we use the offline approximation in which backoff is implemented via epsilon transitions [7]. Prior to decoding, a filtering procedure is used to generate individual sentence-specific WFST language model acceptors for each sentence to be translated. This significantly improves decoding efficiency and is possible because the words which might be postulated in translation are determined by the target language input sentence and the contents of the phrase pair inventory.

1.4. Finite State Grammars for Source Language Subsequences

It often happens that the system is presented with mixed text to translate, for example ASCII characters appearing in Chinese or Arabic text, as in the following example taken from a Chinese-to-English translation task consisting of mixed text extracted from web pages:

此外, 大约三十个摊位也以各类行动电视手机 如 t-dmb (terrestrial digital media broadcasting), s-dmb (satellite digital multimedia broadcasting) 及 dvb-h (digital video broadcasting-handhelds), 提供杜林冬运现场实况转播的画面, 藉以吸引参观者注意。

The source text in such sentences should be ‘translated’ without change, i.e. it should pass through the translation system intact. One solution is to segment the target sentences, translate only the target language portions, and then to form a complete translation by concatenation. However, segmentation is not ideal since it prevents long-span translation and language models from looking across segmentation boundaries. To avoid this problem, a source language acceptor can be included which ensures that the desired segments appear correctly in the translation. For example, suppose two source phrases u_1 and u_2 are found in the target sentence. The acceptor would then accept sequences $V^* \cdot u_1 \cdot V^* \cdot u_2 \cdot V^*$, where V is the source language vocabulary. If degenerate translations for the source phrases are added to the translation and reordering transducer, this acceptor can be included in the translation pipeline as the last step before composition with the English language model. In this way all translations produced (including lattices) have the desired subsequences in the correct order, and all translation scores are based on long-span translation and language model likelihoods. This is a straightforward method to impose many useful constraints in translation, such as ensuring parentheses and quotes are correctly matched, names are correctly transliterated, etc.

1.5. Minimum Error Rate Training

Minimum error rate training under BLEU [8,9] can be used to adjust multiplicative scale factors applied to the component transducers which together make up the TTM. Although only a small number of parameters are adjusted - typically one parameter per component model or distribution - MET can be very effective in tuning systems to domain-specific development sets.

In the systems described here, MET is applied to adjust the lexical language model scale factor, word and phrase insertion penalties, phrase reordering scale factor, phrase insertion scale factor, u -to- v translation model scale factor, v -to- u translation model scale

factor, and three phrase pair count features. The phrase-pair count features track whether each phrase-pair occurred once, twice, or more than twice in the parallel text [10].

MET parameter search procedures as described by Och [9] are now widely used; the only difficulty in apply them to WFSTs is to extract the contribution of each component transducer to the overall translation log likelihood. For this, we use encoded transducers as described by Roark et al. [2,11,12] and implemented in the OpenFST libraries [13].

2. Lattice Rescoring

This section describes lattice rescoring techniques applied to the translation output produced by the first-pass MET baseline system. Apart from MBR (section 2.4) which requires n -best lists, these operations could be applied in first-pass translation; however, we apply these techniques in rescoring subsequent to pruning of the first-pass lattices.

2.1. Large Language Model Rescoring

We apply a second-pass language model that is able to effectively utilise very large quantities of monolingual training text. Large memory and considerable time is required for the estimation of zero cutoff higher-order n -gram language models, typically necessitating partitioning of data and multiple rounds of paired interpolation to produce the final model. An alternative is to build sentence-specific language models. Firstly, counts are gathered for each training text and merged to form a single large counts file. The vocabulary used during the counting process is determined by the set of English words covering the phrases found in the parallel text. There are no cut-offs, so all observed n -grams are included in the model. Sentence-specific counts are obtained by filtering according to the vocabulary of English n -grams in each lattice. The resulting filtered counts are then used to generate sentence-specific language models with “stupid backoff” smoothing [14] in which n -gram scores are defined as

$$S(s_i | s_{i-n+1}^{i-1}) = \begin{cases} \frac{f(s_{i-n+1}^i)}{f(s_{i-n+1}^{i-1})} & \text{if } f(s_{i-n+1}^i) > 0 \\ \alpha S(s_i | s_{i-n+2}^{i-1}) & \text{otherwise} \end{cases} \quad (3)$$

The backoff weight α is the same for each order and the recursion ends with the unigram maximum likelihood estimate.

2.2. Phrasal Segmentation Model Rescoring

Phrasal segmentation models define a mapping from the words of a sentence s_1^I to sequences of translatable phrases u_1^K . Sentences cannot be segmented arbitrarily; the space of possible segmentations is constrained by the contents of the phrase table and contains only those translatable phrases found in the parallel text. We define a probability distribution over phrase sequences and estimate the model parameters from naturally occurring sequences of phrases in a large monolingual source-language training corpus. An order- n phrasal segmentation model assigns a probability to a phrase sequence u_1^K according to

$$P(u_1^K | K, s_1^I) = \prod_{k=1}^K P(u_k | u_1^{k-1}, K, s_1^I) \quad (4)$$

$$\approx \begin{cases} C(K, s_1^I) \prod_{k=1}^K P(u_k | u_{k-n+1}^{k-1}) & \text{if } u_1^K = s_1^I \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

with the additional constraint that each u_k must be a phrase with a known translation. For a fixed s_1^I , the normalisation term $C(K, s_1^I)$ can be calculated. In translation, however, the s_1^I are not fixed so we use the unnormalised likelihoods as scores. The phrase n -gram parameters of Eq. (5) are estimated from the frequencies of occurrence of phrase sequences in the training text. Standard discounting and context-dependent backoff [15] are applied to smooth the maximum likelihood estimates.

The word lattice L produced during first-pass translation is composed with unweighted transducer W to obtain a lattice of phrases ($L \circ W$); this lattice contains phrase sequences and translation scores consistent with the first-pass translation. We now wish to apply the phrase segmentation model distribution of Eq. (5) to this phrase lattice. The conditional probabilities and backoff structure are encoded as weighted finite state acceptor Ψ in the same way as for a regular word language model [7]. The phrasal segmentation model acceptor is then composed with the phrase lattice and projected on the input to obtain the rescored word lattice:

$$L' = (L \circ W) \circ \Psi. \quad (6)$$

The most likely translation after phrasal segmentation model rescoring is given by the path in L' with least cost.

2.3. Model-1 Lattice-to-String Alignment Scores

IBM Model-1 is a simple model of word alignment used in parallel text alignment. Model-1 is not powerful enough to be used alone for translation, but can be used to rank competing translation hypotheses produced by more powerful systems. Introducing a variable a_j which denotes the alignment of t_j in t_1^J to s_{a_j} in s_1^I , the Model-1 alignment distribution is

$$P_{M1}(t_1^J, a_1^J, J | s_1^I) = P_L(J | I) \frac{1}{I^J} \prod_{j=1}^J p_T(t_j | s_{a_j}). \quad (7)$$

The model is such that the maximum likelihood alignment

$$\max_{a_1^J} P_{M1}(t_1^J, a_1^J, J | s_1^I), \quad (8)$$

is readily found via dynamic programming. It is also straightforward to find, for a fixed target sentence t_1^J , the most likely alignment of every translation hypothesis s_1^I in a lattice L , i.e. to simultaneously find the best alignment of every lattice path to the target string. We refer to this as Model-1 lattice-to-string alignment. Of course this could be done by expanding the lattice into a list of distinct hypotheses and aligning each to the target string; however lattice-to-string alignment is faster and retains the compact lattice

representation of hypotheses. However, as discussed by Knight and Al-Onaizan [16], this process cannot be implemented easily with WFSTs. In adding Model-1 alignment scores to the TTM translation lattices, we therefore depart from the WFST formalism and add the Model-1 likelihoods to the TTM lattice scores with non-WFST based lattice-to-string alignment procedures.

2.4. Minimum Bayes Risk Decoding

The final step in translation is Minimum Bayes Risk decoding (MBR) which searches for a hypothesis to minimise the expected loss of translation errors under loss functions that measure translation performance. The rationale is to reconcile estimation criteria (e.g. maximum likelihood) with translation criteria (e.g. BLEU). Since the goal is to maximise the BLEU score, the loss is the negative sentence level BLEU score [17]. Exact computation of statistics needed for BLEU cannot easily be done over lattices, or with finite state approaches, so each translation lattice is expanded into a list of translation hypotheses \mathcal{N} with posterior scores, and the hypothesis is selected which has the least risk relative to the collection of other hypotheses:

$$\hat{\mathbf{S}} = \underset{\mathbf{S} \in \mathcal{N}}{\operatorname{argmin}} \sum_{\mathbf{S}' \in \mathcal{N}} -\text{BLEU}(\mathbf{S}', \mathbf{S}) P(\mathbf{S}' | \mathbf{T}). \quad (9)$$

3. System Development

We describe experiments on the NIST Arabic-English translation task. The development set *mt02-05-tune* is formed from the odd numbered sentences of the NIST MT02 through MT05 evaluation sets; the even numbered sentences form the validation set *mt02-05-test*. Test performance is evaluated using the NIST subsets from the MT06 evaluation: *mt06-nist-nw* for newswire data and *mt06-nist-ng* for newsgroup data. We also report results for the NIST MT08 evaluation. Each set contains four references and BLEU scores are computed for lower-case translations.

The TTM baseline system is trained using all of the available Arabic-English data for the NIST MT08 evaluation. The Arabic text is first morphologically analysed with MADA and words are segmented to separate prefixes [18]. In first-pass translation, decoding proceeds with a 4-gram language model estimated over the parallel text and a 965 million word subset of monolingual data from the English Gigaword Third Edition. Minimum error rate training under BLEU optimises the decoder feature weights using the development set *mt02-05-tune*. In the second pass, a 5-gram zero-cutoff stupid-backoff language model estimated using approximately 4.7 billion words of English newswire text is used to rescore the first-pass lattices. The phrasal segmentation model parameters are trained using a 1.8 billion word subset of the same monolingual training data used to build the second-pass word language model. Further post-processing steps incorporate the Model-1 lattice-to-string alignment scores and MBR.

Table 1. Arabic-English translation results (lower-cased BLEU / TER) for best performing system configuration using phrase pair count features and β_1 probabilities estimated from the alignments.

Method	mt02-05-tune	mt02-05-test	mt06-nist-nw	mt06-nist-ng	mt08-nist
TTM+MET	50.9 / 42.8	50.3 / 43.3	48.1 / 44.3	37.5 / 53.5	43.1 / 49.5
+5g	53.5 / 41.8	52.4 / 42.4	49.6 / 43.9	39.0 / 54.0	43.7 / 49.3
+PSM	53.9 / 42.1	53.3 / 42.7	50.1 / 44.3	39.0 / 54.7	44.3 / 49.3
+MBR	54.0 / 41.7	53.7 / 42.2	51.0 / 43.9	39.4 / 54.1	45.0 / 48.9

3.1. Results and Discussion

Table 1 shows translation performance for each of the various development and evaluation sets as measured by BLEU and TER¹. All of the results in the table were obtained using the MJ1 reordering model with orientation probabilities estimated from alignments. The 1-best output obtained from the lattices after minimum error rate training results in the scores shown in the row labelled ‘TTM+MET’. These lattices are then rescored by each of the post-processing techniques described in section 2, resulting in significant improvements across all sets. While large gains of between 1.5 and 2.7 BLEU points are observed after 5-gram rescoring (row labelled ‘+5g’), phrase segmentation model rescoring results in more modest improvements (row labelled ‘+PSM’). However, these gains are interesting since the models are trained on a subset of the same monolingual data used to train the 5-gram word language model, suggesting that some degree of useful complementary information has been captured by the phrasal segmentation models. The final post-processing step (row labelled ‘+MBR’) shows the results obtained after rescoring the 1000-best list for each sentence using minimum Bayes risk decoding.

In order to demonstrate the advantage of estimating the phrase-specific β_1 reordering probabilities, Table 2 shows translation scores when a flat distribution over all phrase pairs is applied, i.e. the MJ1-Flat reordering model described in section 1.1. These results show that there is a degradation of around 0.4 BLEU points in the MET results, and this degradation is seen throughout the subsequent rescoring steps. A more informed phrase reordering model produces a higher quality MET lattice for rescoring. Therefore, we expect that further improvements in the reordering model will be complementary and benefit even more from our large language model rescoring techniques. However, preliminary experiments with a simplified MJ2 reordering did not yield significant improvements for this Arabic-English translation task and so are not reported here.

Table 2. Arabic-English translation results (lower-cased BLEU / TER) without estimation of the β_1 orientation probabilities for the MJ1 reordering model (MJ1-Flat).

Method	mt02-05-tune	mt02-05-test
TTM+MET	50.4 / 43.3	50.0 / 43.8
+5g	53.0 / 42.2	52.2 / 42.8
+PSM	53.4 / 42.5	53.1 / 43.1

To conclude our analysis of the contribution of each system component, Table 3 shows results obtained when the phrase pair count features are not included in MET. The phrase pair count features clearly contribute significantly to the generation of higher

¹Full MT08 results are available at <http://www.nist.gov/speech/tests/mt/2008/>. It is worth noting that many of the top entries make use of system combination; the results reported here are for single system translations.

Table 3. Translation results (lower-cased BLEU / TER) without phrase pair count features. Two different lattice rescoring orders are compared. On the left, Model-1 (MOD1) rescoring precedes 5-gram and PSM rescoring. On the right, Model-1 rescoring is performed as the final step.

Method	mt02-05-tune	mt02-05-test	Method	mt02-05-tune	mt02-05-test
TTM+MET	48.9 / 43.8	48.6 / 44.1	TTM+MET	48.9 / 43.8	48.6 / 44.1
+MOD1	50.5 / 42.5	50.4 / 43.0	+5g	51.5 / 42.2	51.5 / 42.7
+5g	52.2 / 41.6	52.1 / 42.3	+PSM	52.6 / 42.3	52.6 / 42.7
+PSM	52.9 / 41.9	52.6 / 42.6	+MOD1	53.0 / 41.8	52.6 / 42.5

quality first-pass lattices since there is a degradation of between 1.7 and 2.0 BLEU points with respect to the baseline system.

Table 3 also compares the application of Model-1 rescoring at two different stages in the translation pipeline. Model-1 rescoring proves especially beneficial when directly rescoring the MET lattice (with an improvement of up to 1.8 BLEU points). However, if Model-1 rescoring is applied after 5-gram and phrase segmentation model rescoring there are no real improvements. Two conclusions may be drawn from this. Firstly, that each of the rescoring techniques are a useful source of information when rescoring lattices, and, secondly, applying these techniques sequentially to the same MET lattice does not always provide gains. This suggests it is important to integrate these information sources directly in minimum error rate training prior to generating the lattice.

3.2. Efficiency Considerations

Large-scale statistical machine translation is computationally intensive and an efficient implementation is crucial. To tackle this, we carefully build separate WFSTs that only include the model parameters relevant to each input sentence by prior inspection of input phrases and the general phrase inventory. Using the MJ1 reordering model, the memory required during decoding is less than $\sim 4\text{Gb}$ for most sentences in the reported tasks.

For the longest input sentences, which can exceed 100 words in length, memory requirements may grow beyond this limit and this necessitates pruning. Several pruning strategies may be used, such as standard cost-based pruning for the translation WFST prior to composition with the language model.

However, experience shows that better results are achieved by selecting, for those sentences with a number of states in the translation WFST (prior to language model composition) above a certain threshold, only those phrase segmentations that match the number of phrases in the minimum-number-of-phrases segmentation. This favours segmentations with longer phrases and limits memory requirements without any significant change in translation performance. For *mt02-05-tune* only 39 of the 2075 sentences are affected.

Without further pruning, our system translates the *mt02-05-tune* set (2075 sentences, $\sim 60\text{k}$ words) in a total time of 420 minutes and this can be accomplished in very reasonable time by parallelisation. Figure 3 shows translation time per input word as a function of the sentence length.

As the graph shows, the longer the input sentence, the longer it takes to translate each word. By applying our pruning strategy, we ensure that translation time does not exceed an average rate of 0.43 seconds per word even for the longest sentences. However, around 80% of the sentences are of 40 words or less in length and these are translated with a much quicker rate of 0.30 seconds per word.

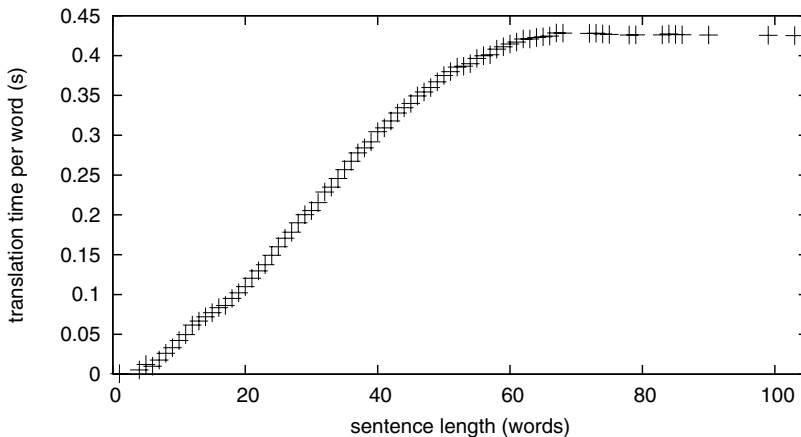


Figure 3. Translation time per word as a function of sentence length for *mt02-05-tune*.

4. Summary and Future Work

We have described the Cambridge University Engineering Department statistical machine translation system that formulates translation as a series of transformations encoded in weighted finite state transducers and decodes using standard finite state operations and algorithms. The system is able to handle very large quantities of data efficiently and effectively and achieves good performance on the 2008 NIST Arabic-English machine translation task, even with the relatively simple MJ1 reordering model.

Future work will investigate whether larger and more consistent gains are possible by integrating the phrasal segmentation models and Model-1 rescoring directly into the MET baseline system. It is also interesting to consider more flexible phrase reordering models by allowing jumps of more than one phrase, although this can lead to a very large search space with many unnecessary hypotheses [2]. One possible solution is to only allow such jumps for a particular list of phrase pairs observed to occur with long-range reorderings in the parallel text from which the phrases are extracted.

Acknowledgements

This work was supported in part under the GALE program of the Defense Advanced Research Projects Agency, Contract No. HR0011-06-C-0022.

References

- [1] Peter F. Brown, Stephen Della Pietra, Vincent J. Della Pietra, and Robert L. Mercer. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2):263–311, 1994.
- [2] Shankar Kumar and William Byrne. Local phrase reordering models for statistical machine translation. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 161–168, 2005.
- [3] Shankar Kumar, Yonggang Deng, and William Byrne. A weighted finite state transducer translation template model for statistical machine translation. *Natural Language Engineering*, 12(1):35–75, 2006.
- [4] Mehryar Mohri, Fernando Pereira, and Michael Riley. Weighted finite-state transducers in speech recognition. In *Computer Speech and Language*, volume 16, pages 69–88, 2002.

- [5] Lambert Mathias and William Byrne. Statistical phrase-based speech translation. In *2006 IEEE International Conference on Acoustics, Speech and Signal Processing*, 2006.
- [6] Christoph Tillmann. A unigram orientation model for statistical machine translation. In *HLT-NAACL 2004: Short Papers*, pages 101–104, Boston, Massachusetts, USA, May 2 - May 7 2004. Association for Computational Linguistics.
- [7] Cyril Allauzen, Mehryar Mohri, and Brian Roark. Generalized algorithms for constructing statistical language models. In *Proceedings of the 41st Meeting of the Association for Computational Linguistics*, pages 557–564, 2003.
- [8] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Meeting of the Association for Computational Linguistics*, pages 311–318, Morristown, NJ, USA, 2001.
- [9] Franz Josef Och. Minimum error rate training in statistical machine translation. In *Proceedings of the 41st Meeting of the Association for Computational Linguistics*, pages 160–167, Morristown, NJ, USA, 2003.
- [10] Oliver Bender, Evgeny Matusov, Stefan Hahn, Sasa Hasan, Shahram Khadivi, and Hermann Ney. The RWTH Arabic-to-English spoken language translation system. In *Proceedings of the 2007 Automatic Speech Understanding Workshop*, pages 396–401, 2007.
- [11] Brian Roark, Murat Saraclar, and Michael Collins. Discriminative n-gram language modeling. *Computer Speech and Language*, 21(2):373–392, 2007.
- [12] Lambert Mathias. *Statistical Machine Translation and Automatic Speech Recognition under Uncertainty*. Dissertation, Johns Hopkins University, 2007.
- [13] Cyril Allauzen, Michael Riley, Johan Schalkwyk, Wojciech Skut, and Mehryar Mohri. OpenFST: a general and efficient weighted finite-state transducer library. In *Proceedings of the 9th International Conference on Implementation and Application of Automata*, pages 11–23. Springer, 2007.
- [14] Thorsten Brants, Ashok C. Popat, Peng Xu, Franz J. Och, and Jeffrey Dean. Large language models in machine translation. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 858–867, 2007.
- [15] Stanley F. Chen and Joshua Goodman. An empirical study of smoothing techniques for language modeling. In Arivind Joshi and Martha Palmer, editors, *Proceedings of the 34th Meeting of the Association for Computational Linguistics*, pages 310–318, San Francisco, 1996. Morgan Kaufmann Publishers.
- [16] Kevin Knight and Yaser Al-Onaizan. Translation with finite-state devices. In *AMTA '98: Machine Translation and the Information Soup: Third Conference of the Association for Machine Translation in the Americas*, pages 421–437, London, UK, 1998. Springer-Verlag.
- [17] Shankar Kumar and William Byrne. Minimum Bayes-risk decoding for statistical machine translation. In *HLT-NAACL 2004*, Boston, Massachusetts, USA, May 2 - May 7 2004. Association for Computational Linguistics.
- [18] Nizar Habash and Owen Rambow. Arabic tokenization, part-of-speech tagging and morphological disambiguation in one fell swoop. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 573–580, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics.

Proper Noun Recognition and Classification Using Weighted Finite State Transducers

Jörg DIDAKOWSKI ^a, Marko DROTSCHMANN ^a

^a *Berlin-Brandenburgische Akademie der Wissenschaften, Jägerstr. 22/23, 10117
 Berlin, Germany*

Abstract. This paper presents a new approach to proper noun recognition and classification in which the knowledge of ambiguities within morphological analyses is used exhaustively in the analysis. Here a proper noun recognizer/classifier is defined by proper noun context patterns on the one hand and by a filter that takes the ambiguity information into account on the other hand. Furthermore, techniques like a lemma based coreference resolution or the softening of the closed world assumption made by the morphology are presented which improve the analysis. The approach is implemented by weighted finite state transducers and tested within the analysis system SynCoP via a hand-written grammar.

Keywords. information extraction, named entity recognition, finite state machines, coreference resolution, tagging, constraints

Introduction

Proper noun recognition and classification is, as a subtask of named entity recognition (NER), a fundamental task in information extraction (IE). Our interest in the proper noun recognition and classification lies in the annotation of large German text corpora for exploitation purposes. In corpus linguistics exploitation can be facilitated by concordances presenting a word in its different contexts. Here part-of-speech or deeper linguistic information, for example syntactic functions, is used to restrict the set of concordance lines (see [1]). A second way to exploit corpora are syntactic collocation databases where semantic connections can be discovered by highly frequent co-occurrences of words in special syntactic constructions (see [2] and [3]). To realize any of the mentioned techniques the corpora have to be syntactically annotated. Furthermore, the annotation has to be done fully automatically in the case of very large corpora where manual annotation is not practicable any more.

In German as well as in English, there exist many problems which make the recognition and classification of proper nouns difficult: One problem is caused by structural ambiguity such as PP-attachment ambiguity and ambiguity of conjunction scope. Another problem is caused by semantic ambiguities in terms of metonymy where a proper noun can belong to several proper noun classes. Furthermore, proper nouns are an open class and very productive so that, in principal, every word in a general-language lexi-

con can become a proper noun (see [4]). In addition, in German some language specific characteristics complicate the proper noun recognition in comparison to English: proper nouns and common nouns can't be distinguished by the uppercase letter. Moreover, the free word order complicates the detection of the scope of proper nouns (see [5]).

This paper presents a method based on weighted finite state transducers which tackle the problem of proper noun recognition in German. The approach is based on [6] where proper noun recognition is treated as a disambiguation problem. Proper noun contexts are marked by brackets and additionally tags are used to mark proper noun context parts. In the approach information about the morphological ambiguity of proper nouns is used in the analysis. If, for example, a proper noun is categorically unambiguous and if it has a definite proper noun class it is in all probability a proper noun of the corresponding class. By contrast, if a proper noun is ambiguous the reliability of being a proper noun depends gradually on the ambiguity. In our approach the information about morphological ambiguity is also used in conjunction with human nouns, company nouns and location nouns which indicate proper nouns. We present new techniques to handle this reliability information in connection with contextual and coreferential information. Furthermore, in the approach scope ambiguities and reliability of proper noun classification are covered by linguistic criteria formalized by an idempotent semiring.

The paper is organized as follows: Section 1 gives basic definitions and notations. Section 2 describes the used representation of the input and the analyses. In section 3 the basic idea of marking and classifying proper nouns is presented. After this, section 4 focuses on the techniques of writing a proper noun context grammar. The approach is extended in section 5 by the softening of the closed world assumption made by the morphology, in section 6 by coreference resolution and in section 7 by a global filter which implements some generalizations. Finally, the approach is implemented and tested with the analysis system SynCoP in section 8.

1. Definitions and Notations

In our approach analyses are generated and scored over an input by a WFST such that they can be judged by linguistic criteria. A weighted finite state transducer $T = (\Sigma, \Delta, Q, q_0, F, E, \lambda, \rho)$ over a semiring S is an 8-tuple such that Σ is the finite input alphabet, Δ is the finite output alphabet, Q is the finite set of states, $q_0 \in Q$ is the start state, $F \subseteq Q$ is the set of final states, $E \subseteq Q \times (\Sigma \cup \epsilon) \times (\Delta \cup \epsilon) \times S \times Q$ is the set of transitions, λ is the initial weight and $\rho : F \mapsto S$ is the final weight function mapping final states to elements in S .

The input is represented as an acyclic finite state automaton. The application of an analysis transducer (ANALYZER) is done by composition of the input (INPUT) in form of an identity transducer with the analysis transducer. Then the second projection of the resulting transducer is taken which contains the scored analyses (ANALYSIS):

$$\text{ANALYSIS} =_{def} \text{Range}(\text{INPUT} \circ \text{ANALYZER}) \quad (1)$$

Here the regular expression notation of [7] (slightly extended) is used.¹

¹See appendix for regular expression notation details. Here the precedence is defined top down. The distinction between the automaton A and the identity transducer that maps every string of A to itself is ignored.

Scope ambiguities and the reliability of proper noun recognition and classification are covered by linguistic criteria. With these criteria it is possible to compare analyses by means of scores depending on the particular linguistic criterion. The linguistic criteria are formalized by the notion of a semiring ([8]). Let $S \neq \emptyset$ be a set and \oplus (called addition) and \otimes (called multiplication) binary operations on S , then $(S, \oplus, \otimes, \bar{0}, \bar{1})$ is called a semiring if $(S, \oplus, \bar{0})$ is a commutative monoid, $(S, \otimes, \bar{1})$ is a monoid and \otimes distributes over \oplus . Linguistic criteria are represented by this structure. To judge analyses via addition an additive idempotent semiring has to be used to create a partial order over S . Thus a partial order is defined by $(a \leq_S b) \Leftrightarrow (a \oplus b = a)$. Here $a \leq_S b$ means that a is “better” than b with respect to linguistic criteria.

It will be necessary to judge analyses by more than one linguistic criterion; therefore, the criteria are ranked by preference. To model this the *composition* of idempotent semirings is defined as follows ([8]): if a linguistic preference $(S_1, \oplus_1, \otimes_1, \bar{0}_1, \bar{1}_1) \succ (S_2, \oplus_2, \otimes_2, \bar{0}_2, \bar{1}_2) \succ \dots \succ (S_n, \oplus_n, \otimes_n, \bar{0}_n, \bar{1}_n)$ is given and if for each semiring a partial order is defined by \oplus , then the composition $(S, \oplus, \otimes, \bar{0}, \bar{1}) = (S_1, \oplus_1, \otimes_1, \bar{0}_1, \bar{1}_1) \circ (S_2, \oplus_2, \otimes_2, \bar{0}_2, \bar{1}_2) \circ \dots \circ (S_n, \oplus_n, \otimes_n, \bar{0}_n, \bar{1}_n)$ is the vectorization of the individual domains and of the operation \otimes . This corresponds to the *crossproduct* of semirings (cf. [9]). The operation \oplus which compares analyses is defined in a special way, if $(a_1, a_2, \dots, a_n) \in S$ and $(b_1, b_2, \dots, b_n) \in S$ are given:

$$(a_1, a_2, \dots, a_n) \oplus (b_1, b_2, \dots, b_n) = \begin{cases} (a_1, a_2, \dots, a_n) & \text{if } (a_1, a_2, \dots, a_n) = (b_1, b_2, \dots, b_n) \\ (a_1, a_2, \dots, a_n) & \text{if } a_1 = b_1 \text{ and } a_2 = b_2 \text{ and } \dots \text{ and } a_{k-1} = b_{k-1} \\ & \text{and } a_k \oplus_k b_k = a_k \\ & \text{with } k \leq n \text{ and } a_k \neq b_k \\ (b_1, b_2, \dots, b_n) & \text{if } a_1 = b_1 \text{ and } a_2 = b_2 \text{ and } \dots \text{ and } a_{k-1} = b_{k-1} \\ & \text{and } a_k \oplus_k b_k = b_k \\ & \text{with } k \leq n \text{ and } a_k \neq b_k \end{cases} \quad (2)$$

The resulting semiring is now idempotent as well and a partial order can be defined by \oplus .

Extracting the most likely analyses with respect to linguistic criteria in a WFST T which is the result of the application of an analysis transducer is a classical best-path problem. Weights along a path of T are combined by the abstract multiplication and create costs. If several paths are in T their weight equals the abstract addition of weights of the different paths, that means the “best” cost (see [10]). The most likely analyses are simply represented by paths causing these “best” costs.

In the following examples the STTS (Stuttgart/Tübinger Tagset, a tagset for German) is used.

2. Representation of the Input and the Analyses

In this section the representation of the input and the analyses are defined. The raw input text is enriched with morphological information. Here the enriched input consists of sequences of word boundaries, lemmata and categories. Such sequences can be represented

compactly by an acyclic automaton and can generally be described by the following regular expression:²

$$\{\@\@\} [\{\@\}\text{LEM CATEGORY}]*\{\@\@\} \quad (3)$$

The tag $\{\@\@\}$ surrounds the text, the tag $\{\@\}$ indicates a word boundary, the expression LEM denotes all possible lemma forms and the expression CATEGORY denotes all possible categories. Such a text automaton contains all possible morphological readings of a text which are usually highly ambiguous (cf. [11]). As a concrete example the regular expression for the German sentence “Petra sieht die Stadt Burg.” (“Petra saw the city Burg.”) is given. Here, for lack of space only one possible reading is covered and some POS features like noun case or noun gender as well as the word boundary feature Case for the word surface form are left out:

$$\begin{array}{llll} \{\@\@\} & & & \\ \{\@ \text{ CAmb=no PnAmb=0 InAmb=-} \} & \text{Petra} & \{\text{NE Nametype=firstname}\} & \\ \{\@ \text{ CAmb=no PnAmb=- InAmb=-} \} & \text{sehen} & \{\text{VVFIn}\} & \\ \{\@ \text{ CAmb=yes PnAmb=- InAmb=-} \} & \text{die} & \{\text{ART}\} & \\ \{\@ \text{ CAmb=no PnAmb=- InAmb=0} \} & \text{Stadt} & \{\text{NN SemClass=location}\} & \\ \{\@ \text{ CAmb=yes PnAmb=4 InAmb=3} \} & \text{Burg} & \{\text{NN SemClass=location}\} & \\ \{\@ \text{ CAmb=no PnAmb=- InAmb=-} \} & . & \{\text{SYMBOL Type=punct}\} & \\ \{\@\@\} & & & \end{array} \quad (4)$$

In the example, the noun feature *SemClass* gives semantic information. With this information it can be decided whether a noun is a proper noun indicator or not and what type of proper noun indicator a noun is (location noun, organization noun, person noun). The proper noun feature *Nametype* gives information about the proper noun class (location name, organization name, etc.) or about proper noun subclasses (first name, last name). The tag $\{\@\}$ is defined as a complex category and its features contain several kinds of information: The feature *Case* gives information about the surface form of the specific word, whether it is upper or lower case, etc. The other features give information about the ambiguity of the morphological analyses (here the feature value “-” stands for undefined):

- **CAmb:** This feature gives information about whether the analysis is categorical ambiguous (yes) or not (no).
- **PnAmb:** This feature gives information about the strength of ambiguity with respect to proper nouns in a range from zero to five: 0) there are no relevant ambiguities with respect to a proper noun; 1) an ambiguity exists within a proper noun class (e.g. first name vs. last name); 2) an ambiguity exists between proper noun classes; 3) an ambiguity exists between proper noun and proper noun indicator (human noun, location noun or company noun); 4) an ambiguity exists between proper noun and noun; 5) an ambiguity exists between proper noun and a non-noun category.
- **InAmb:** This feature gives information about the strength of ambiguity with respect to the proper noun indicators human noun, location noun and company noun in a range from zero to four: 0) there are no relevant ambiguities with respect to a proper noun indicator; 1) an ambiguity exists within proper noun indicator

²The curly brackets denote possibly underspecified categories. The features are defined with respect to an inheritance hierarchy and are represented as transition labels. Underspecification is realized as the disjunction of all maximal subtypes of a super type.

classes (human, location or company); 2) an ambiguity exists between proper noun indicator and other semantic noun classes; 3) an ambiguity exists between proper noun indicator and proper noun; 4) an ambiguity exists between proper noun indicator and a non-noun category.

The example sentence contains two proper nouns, the person name “Petra” and the location name “Burg”. The word “Petra” is definitely a proper noun, because it is not categorically ambiguous (CAmb=no). The word “Burg”, however, is categorically ambiguous between a location noun and a proper noun. Therefore it receives the corresponding ambiguity information (CAmb=yes PnAmb=4 InAmb=3). Via the proper noun indicator “Stadt” the word “Burg” can be identified as a location noun:

```
{@@}
{PN}
  {@}  Petra  {NE Nametype=firstname}  {@FirstName Reli=4}
{/PN Class=person}
{@}    sehen  {VVFIN}
{PN}
  {@}    die    {ART}
  {@}    Stadt  {NN SemClass=geo}      {@GeoNoun}
  {@}    Burg   {NE Nametype=geoname}  {@GeoName Reli=2}
{/PN Class=location}
{@}    .        {SYMBOL Type=punct}
{@@}
```

(5)

For the sake of readability the features of the word boundaries are left out in the example analysis. The proper noun contexts are marked by the tags {PN} and {/PN}. Within these contexts parts like proper nouns or proper noun indicators are marked by syntactic/semantic tags (e.g. {@FirstName}, {@Geonoun}). Here the feature Reli gives information about the reliability of the classification in a range from zero to five. Furthermore, the proper noun contexts are classified by their proper noun class via the feature Class (e.g. location, person).

3. Marking of Proper Noun Contexts

The marking of proper noun contexts is treated as a chunking problem. We follow the approach in [8], in which chunking is performed by the *optional insertion operator*. The optional insertion operator is defined by the following regular expression, if GRAMMAR denotes the proper noun contexts and if P and S denote the brackets:

$$\text{GRAMMAR}(\rightarrow)P \dots S =_{def} [?^*[0.x.P]\text{GRAMMAR}[0.x.S]]?^* \quad (6)$$

This operator optionally brackets the proper noun contexts specified by the regular expression GRAMMAR with the brackets P and S in all possible variations. This operator is defined without any complementation operations. Thus, transductions can be performed and tags can be inserted via epsilon.

Proper noun contexts should only be marked if a corresponding reliability is given. Furthermore, analyses with the highest degree of reliability should be preferred. This principle is formalized by the *reliability optimization criterion*. This criterion is formalized by the max semiring $(\mathbb{R} \cup \{-\infty\}, \max, +, -\infty, 0)$. Here negative numbers are as-

signed to patterns to model gradual unreliability. Positive numbers are assigned to patterns to model gradual reliability.

The linguistic criteria *chunk inclusiveness* and *chunk connectedness* are defined to implement a longest match strategy.³ The criteria express that words should belong to chunks and that words should form large chunks. The chunk connectedness criterion and the chunk inclusiveness criterion are formalized separately by the tropical semiring $(\mathbb{R} \cup \{\infty\}, \min, +, \infty, 0)$. The criteria are implemented by assigning negative numbers to symbols within chunks on the one hand and by assigning positive numbers to bracket pairs on the other hand.

All three of the mentioned linguistic criteria are ranked as follows: reliability optimization \succ chunk connectedness \succ chunk inclusiveness. The reliability optimization criterion is ranked higher than the longest match criteria. That is, the longest match is used only for extension disambiguation purposes. As the proper noun contexts should only be marked if a corresponding reliability is given, the chunk connectedness criterion is ranked higher than the chunk inclusiveness criterion. The ranking is achieved by composition of idempotent semirings.

The optional insertion operator is now adjusted as follows:

$$\text{GRAMMAR}(\rightarrow)P \dots S \stackrel{\text{def}}{=} [?^* [0.x.P] [\text{GRAMMAR} \circ [? \langle 0, 0, -1 \rangle]^*] [0.x.S] \langle 0, 1, 0 \rangle]^* ?^* \quad (7)$$

The proper noun context chunkers are independently built for each proper noun class by the optional insertion operator. Here, the proper noun classification is performed via the chunk brackets. The several chunkers are combined by union. Then the star closure is applied. The chunkers compete with each other in this definition. So the analysis transducer (ANALYZER) is defined by the following regular expression:

$$\begin{aligned} \text{ANALYZER} \stackrel{\text{def}}{=} & [[\text{GRAMMAR}_1(\rightarrow)P_{1_} S_1] \mid \\ & [\text{GRAMMAR}_2(\rightarrow)P_{2_} S_2] \mid \\ & \dots \mid \\ & [\text{GRAMMAR}_n(\rightarrow)P_{n_} S_n]]^* \end{aligned} \quad (8)$$

4. Writing a Grammar for Proper Noun Contexts

The patterns which describe proper noun contexts and which are used in the construction of the chunker are defined in terms of a grammar. The task of writing a grammar is split up into two parts: 1) the design of patterns which describe all possible proper noun contexts with all possible markings of parts (PATTERNS), 2) the design of a filter which restrict and score the patterns (FILTER). Both parts, PATTERNS and FILTER, are independently implemented and are combined by composition:

$$\text{GRAMMAR} \stackrel{\text{def}}{=} \text{PATTERNS} \circ \text{FILTER} \quad (9)$$

The patterns (PATTERNS) model possible proper noun contexts. Here, syntactic and/or semantic tags are inserted to indicate the parts of the proper noun contexts

³The basic idea of implementing a longest match constraint using the weights of a WFST is presented in [12].

(@GeoName, @GeoNoun, etc.). In this phase the ambiguity information of the input is ignored. The patterns are modeled as if the input is definite. An example regular expression for a simple person name pattern will be given:

$$\begin{aligned}
 & \text{HUMANNOUN} =_{def} \\
 & \{ \{ @ \} \text{LEM} \{ \text{NN SemClass} = \text{human} \} [0.x. \{ @ \text{HumanNoun} \}] \\
 & \text{FIRSTNAME} =_{def} \\
 & \{ \{ @ \} \text{LEM} \{ \text{NE Type} = \text{firstname} \} [0.x. \{ @ \text{FirstName Reli} = 0 \}] \\
 & \text{LASTNAME} =_{def} \\
 & \{ \{ @ \} \text{LEM} \{ \text{NE Type} = \text{lastname} \} [0.x. \{ @ \text{LastName Reli} = 0 \}] \\
 & \text{PATTERN} =_{def} \\
 & \text{HUMANNOUN FIRSTNAME}^* \text{LASTNAME}
 \end{aligned} \tag{10}$$

Via the regular expression PATTERN occurrences of last names (LASTNAME) optionally preceded by first names (FIRSTNAME) are marked if they are preceded by a human noun (HUMANNOUN). Here the first names are labeled by the tag { @FirstName }, the last name is labeled by the tag { @LastName } and the human noun is labeled by the tag { @HumanNoun }; the tags are inserted via epsilon. Here the reliability of the labeling is initialized by zero (Reli=0) which indicates potential proper nouns. Such patterns can be combined by union.

The filter (FILTER) takes the word boundary information into account. Here the filter restricts and scores the proper noun context patterns via the ambiguity information. The filter can consist of several filters competing with each other. Thus, several filters are combined by union. A filter is defined by constraints. A constraint can be used to restrict patterns, to score patterns or to rewrite information. The constraints that restrict patterns are implemented via the exist operator. It can be postulated that every proper noun context pattern contains at least one specific pattern A: \$A. In contrast, it can also be postulated that all proper noun context patterns do not contain a specific pattern A: ~ \$A. This means, it is possible to implement the necessity of a special degree of reliability of proper noun recognition and classification. The constraints which score or rewrite patterns are implemented by the *score and rewrite operator*, which is defined by the following regular expression (cf. [8]):

$$A \Rightarrow_{\omega} _ =_{def} [\sim \$ [\text{Dom}(A) - []] A(\omega, 0, 0)^* \sim \$ [\text{Dom}(A) - []] \tag{11}$$

Transductions defined by the regular expression A are performed locally on the proper noun context patterns. Additionally scores can be assigned by ω with respect to the reliability optimization criterion. To illustrate the different types of constraints some example constraints will be given:

$$\begin{aligned}
 \text{CONSTRAINT}_1 &=_{def} \sim \$ [\{ @ \text{PnAmb} = 5 \} \text{LEM} \{ \text{NE} \}] \\
 \text{CONSTRAINT}_2 &=_{def} \$ [\{ @ \text{PnAmb} = 0 \} \text{LEM} \{ \text{NE} \}] \\
 \text{CONSTRAINT}_3 &=_{def} \{ @ \text{FirstName Reli} = 0 : 3 \} \Rightarrow_{10} _
 \end{aligned} \tag{12}$$

The constraint CONSTRAINT₁ forbids proper nouns with an ambiguity degree of 5. The constraint CONSTRAINT₂ asks for a non-ambiguous core. Finally, via the constraint CONSTRAINT₃ the reliability information of the tag { @FirstName } is changed from 0 to 3 and the score 10 is assigned. Several constraints can be combined to a filter by composition. Here the order of the constraints matter.

5. Softening the Closed World Assumption Made by the Morphology

For each word a morphology returns a set of possible analyses. Here some words are analyzed as a proper noun, others are not. So the morphology prescribes whether a word can be a proper noun or not. Via the morphology a *closed world assumption* is presumed. But almost every word in a general-language lexicon can become a proper noun. Therefore, it should be possible to change the morphological information. Thus, the definition of a proper noun context grammar is extended by a POS expansion part (EXPANSION):

$$\text{GRAMMAR} =_{def} \text{EXPANSION} . \text{o} . \text{PATTERNS} . \text{o} . \text{FILTER} \quad (13)$$

Within the expansion part the non-NE POS tags of uppercase words are optionally mapped to the NE tag. Furthermore, proper noun classes or proper noun subclasses are optionally mapped to other proper noun classes or subclasses. Here the transformations have several degrees of strength. That is to say, it is easier to transfer a proper noun class into another than to transform a non-NE POS into a proper noun. The transformation information can be used in the filter definition of a grammar.

We use the word boundary feature *Trans* to indicate the strength of transformation. The degree of strength ranges from zero to five: 0) no transformation, 1) transform a proper noun subclass into another proper noun subclass, 2) transform a proper noun class into another proper noun class, 3) transform an unknown word into a proper noun, 4) transform a noun into a proper noun, 5) transform all other POS into a proper noun.

The feature *Trans* is initialized with zero in the input. An example for an expansion transducer which optionally transforms nouns into proper nouns is given below. Note that only nouns were transformed which are not homographic to a proper noun:

$$\begin{aligned} \text{EXPANSION} =_{def} \\ [?*\{ @ \text{PnAmb} = - \text{Trans} = 0 : 4 \} [\{ \text{NN} \} . \text{x} . \{ \text{NE} \}]] *? * \end{aligned} \quad (14)$$

6. Lemma Based Coreference Resolution

Within newspaper texts proper nouns are usually introduced in some specific way (e.g. via indicator nouns like president, city, company). After the introduction the proper nouns can be used without any extra specifications. So, in some contexts proper nouns can be detected very easily. In other contexts the detection is harder. We overcome this problem by a coreference resolution: a lemma which is once classified as a sure proper noun is a proper noun in the whole text.

In our approach we distinguish between potential and sure proper nouns; sure proper nouns are used to support potential proper nouns by scores. More precisely, potential proper nouns should not be marked at all without coreference support. For this purpose we distinguish between a filter for sure proper nouns ($\text{filter}_{\text{sure}}$) and for potential proper nouns ($\text{filter}_{\text{potential}}$). Here $\text{filter}_{\text{sure}}$ is defined as in section 4 and defines the reliability of classification of proper nouns. By contrast, $\text{filter}_{\text{potential}}$ punishes syntactic/semantic tags which mark proper nouns with the feature-value pair $\text{Reli} = 0$ by a negative number with respect to the reliability optimization criterion. Note that the potential proper nouns are initially marked by this feature-value pair. Thus, the marking of the individual potential proper nouns is suppressed. Now the construction of a proper noun context grammar is adjusted as follows:

$$\text{GRAMMAR} =_{def} \text{EXPANSION} . \circ . \left[\begin{array}{l} [\text{PATTERNS} . \circ . \text{FILTER}_{sure}] \\ [\text{PATTERNS} . \circ . \text{FILTER}_{potential}] \end{array} \right] \quad (15)$$

To extract the sure proper nouns an extraction transducer can be built. An example for an extraction transducer for sure last names is given by the following regular expression:

$$\text{EXTRACT_LASTNAME} =_{def} [\{@\}.x.\{@\}]\text{LEM}[\{NE\}.x.\{NE\}]\{@Last\text{Name Reli}=1:0\} \quad (16)$$

$$\text{EXTRACTOR} =_{def} [?:0]^* \text{EXTRACT_LASTNAME} \langle 1, 0, 0 \rangle [?:0]^* \quad (17)$$

With this transducer all word information analyzed as sure last name with a reliability of one are extracted from the analysis and scored by the weight $\langle 1, 0, 0 \rangle$. The reliability information is rewritten to zero for mapping purposes. Such extraction transducers can be built accordingly for other syntactic/semantic functions with other degrees of reliability (here various scores can be assigned). The various extraction transducers can be combined by union.

After the application of the analysis transducer all sure and potential proper nouns are marked in the analysis. Now the analysis is split up into two variants via a best path search: 1) the analysis contains the support of the best readings (ANALYSIS_{best}), 2) the analysis contains all scored readings (ANALYSIS_{all}). After the extraction of the sure proper nouns, the potential proper nouns are supported via these sure proper nouns:

$$\text{ANALYSIS} =_{def} \text{ANALYSIS}_{all} \ \& \ [?*Range(\text{ANALYSIS}_{best} . \circ . \text{EXTRACTOR})]^*?^* \quad (18)$$

7. The Use of a Global Filter

Some generalities can be covered by a filter that globally works on the input and performs special rewritings of information (e.g. rewritings of word boundary features). The changed information can be used in the grammar. So, the analysis transducer is extended by a global filter denoted by `GLOBALFILTER`:

$$\text{ANALYZER}' =_{def} \text{GLOBALFILTER} . \circ . \text{ANALYZER} \quad (19)$$

In German function words or verbs etc. generally are only capitalized in the beginning of a sentence. Here, homography to a proper noun can only occur in such positions. This information should be integrated into the word border information. For this purpose the word boundary feature `SInit` is defined. The feature indicates whether a word is sentence initial (yes) or not (no). In the input the feature is initialized with `no`. Whenever a potential sentence initial position is detected the following word border feature `SInit` is changed to `yes`. In this connection, a global filter can be defined by a regular expression as follows whereas the score and rewrite operator is used:

$$\text{GLOBALFILTER} =_{def} [\{\text{SYMBOL Type=punct}\}|\{@@\}]\{@ \text{SInit=no:yes}\} \Rightarrow_0 _ \quad (20)$$

8. Testing and Results

The approach presented in this paper is implemented with the Syntactic Constraint Parser (SynCoP), which is based on WFSTs (see also [8], [6] and [2]). SynCoP consists of a grammar compiler, a grammar-driven parser, and a preprocessing module which comprises tokenizing and the recognition of multi-word units. The engine admits specification of the parser along with the preprocessing module by means of a grammar written in XML. Thus the engine can be easily adapted to individual conceptions of analysis.

The morphological analysis is performed by the TAGH morphology ([13]). The TAGH morphology is a complete finite state morphology which analyses productive German derivation and composition. It uses semantic noun classes taken from *LexikoNet* [14] allowing the detection of indication nouns.

Our hand-written grammar covers person, location and organization name contexts. We mark proper nouns, proper noun indicators, titles and similar name affixes such as organization markers, and special types of verbs (for example communication verbs). Multi-word units with respect to VIPs, company and geographical names are handled in the preprocessing module. The *Reli* feature which marks the degree of reliability of contexts and which helps improving the grammar was ignored during the evaluation process.

To evaluate the grammar, a test corpus consisting of 100 annotated newspaper articles in the field of politics was used. The corpus contained 54,998 words (67,228 tokens respectively) of which 3,068 proper nouns consisting of 3,938 words were annotated: 1214 of them were person names, 1009 were location names and 845 were organization names. 5,224 words were analyzed by the TAGH morphology as a possible proper noun. 27.6% of them had an additional analysis as a non-noun category, 52.9% were ambiguous between a proper noun and a noun and 12.8% were ambiguous between several proper noun classes. In addition, 0.7% of the words could not be analyzed by the morphology.

The accuracy of the detection and classification of proper nouns seems quite promising. The precision approximately reaches the following values: 94% for person names, 93% for location names, 94% for organization names. The recall approximately reaches the following values: 93% for person names, 84% for location names, 80% for organization names. The precision and recall values are comparable to other rule-based approaches (see [15] or [16]). The TAGH morphology has a broad coverage for political texts. This leads to generally complete morphological analyses, especially with respect to ambiguity, that the presented approach can use in order to improve the results. As the morphology is able to handle compounds, semantic classes can be assigned to arbitrarily complex words. This represents another advantage the approach can use to more reliably recognize and classify proper nouns in German texts. Furthermore, the evaluation results show that person names are more intensively introduced in the corpus, thus the recall is that much better for this proper noun class.

9. Conclusion and Future Work

In this paper, a new method for proper noun recognition and classification, which is implemented by weighted finite state transducers, has been presented. Via an excessive use of ambiguity information, the inclusion of a lemma based coreference resolution and

the softening of the closed world assumption made by the morphology quite promising results are achieved.

We are planning to combine the proper noun recognition and classification with a syntactic dependency parser implemented by SynCoP (see [8]) in the framework of weighted finite state transducers. We hope that both parts benefit by this connection. Via this combination, we aim to use the proper noun recognition and classification in the construction of the German collocation database *word profile* ([2]). Here we are very interested in the effects of this integration.

10. Appendix:Notations

$\sim A$	complement
$\$A$	contains (all strings containing at least one A)
A^*	Kleene star
$A \ B$	concatenation
$A \mid B$	union
$A \ \& \ B$	intersection
$A \ .x. \ B$	crossproduct
$A \ .o. \ B$	composition
$Dom(A)$	the domain of a rational transduction
$Range(A)$	the range of a rational transduction
$\{A \ feat_1=val_{1x} \dots feat_n=val_{nx}\}$	category A with specified features
$feat_x=val_{x1} : val_{x2}$	mapping of feature values
$?$	sigma
$?^*$	sigma star
ϵ	epsilon
$[\]$	the empty string language
$\langle \omega \rangle$	weight
$[\text{ and }]$	square brackets that group expressions

References

- [1] Pasi Tapanainen and Timo Järvinen. Syntactic concordances. In *Corpora Galore - Analyses and Techniques in Describing English*. Amsterdam/Atlanta:GA:Rodopi, 2000.
- [2] Alexander Geyken, Jørg Didakowski, and Alexander Siebert. Generation of word profiles on the basis of a large balanced german corpus. In *Proceedings of EURALEX 2008*, 2008.
- [3] A. Kilgariff, P. Rychly, P. Smrz, and D. Tugwell. The sketch engine. In *Proceedings of EURALEX 2004*, pages 105–116, 2004.
- [4] Hayssam Trabousli. A local grammar for proper names. Thesis University of Surrey, 2004.
- [5] Christine Thielen. An approach to proper name tagging for german. In *Proceedings of ACL SIGDAT-95*, 1995.
- [6] Jörg Didakowski, Alexander Geyken, and Thomas Hanneforth. Eigennamenerkennung zwischen morphologischer analyse und part-of-speech tagging: ein automatentheoriebasierter ansatz. *Zeitschrift für Sprachwissenschaft* 26, pages 157–186, 2007.
- [7] Lauri Karttunen. The replace operator. In *Proceedings of ACL-95*, pages 16–23, 1995.
- [8] Jørg Didakowski. Syncop - combining syntactic tagging with chunking using weighted finite state transducers. In *Proceedings of FSMNLP 07*, 2007.
- [9] Udo Hebisch and Hanns J. Weinert. *Halbringe*. Stuttgart:Teubner, 1993.
- [10] Mehryar Mohri. Semiring frameworks and algorithms for shortest-distance problems. *Languages and Combinatorics*, 7(3):321–350, 2002.

- [11] Kimmo Koskenniemi. Finite-state parsing and disambiguation. In *Proceedings of COLING-90*, volume 2, pages 229–232, 1990.
- [12] Thomas Hanneforth. Longest-match pattern matching with weighted finite state automata. In *Proceedings of FSMNLP 05*, 2005.
- [13] Alexander Geyken and Thomas Hanneforth. Tagh: a complete morphology for german based on weighted finite state automaton. In *Proceedings of FSMNLP 05*, 2005.
- [14] A. Geyken and N. Schrader. Lexikonet - a lexical database based on type and role hierarchies. In *Proceedings of LREC-06*, 2006.
- [15] G. Neumann and J. Piskorski. A shallow text processing core engine. Technical Report DFKI, 2002.
- [16] M. Volk and S. Clematide. Learn - filter - apply - forget. mixed approaches to named entity recognition. In *NLDB'01*, pages 153–163, 2001.

Finite-State Local Grammars for Disambiguating Conjunctions in Portuguese Proper Names

Samuel ELEUTÉRIO ^{a,1} Elisabete RANCHHOD ^{b,2}

^a *Instituto Superior Técnico*

^b *University of Lisbon*

Abstract. Like common noun phrases, proper names contain ambiguous conjoined phrases that make their delimitation and classification difficult in text. This paper presents a finite-state approach to the disambiguation of Portuguese candidate proper name strings containing the coordinating conjunction *e* (and). In such name strings, the conjunction can denote a relation between two independent names, but it can also be part of a multiword proper name. The coordination of multiword independent names may involve ellipsis of some lexical constituents, which causes additional difficulties to proper name identification and classification.

Keywords. Portuguese named entity recognition, ambiguous proper names, information extraction, finite-state grammar, local grammar, coordination and ellipsis.

Introduction

This paper presents an experiment aiming to resolve ambiguity caused by the coordinating conjunction *e* (and) in Portuguese named entity recognition, and describes the heuristics used to disambiguate candidate proper name strings containing that conjunction. We are interested in naming expressions referring to entities as *persons* (given and family names, titles, etc.), *organizations* (corporations, public and governmental institutions, etc.) and *locations* (regions, countries, cities, mountains, rivers, etc.).

As well as in other Romance languages (see [1] for French, [2] for Spanish), in Portuguese the formal criterion more operational for proper names, that distinguishes them from common nouns, is the initial capital letter. In fact, in Portuguese texts, the use of initial capitalized words in the interior of a sentence, i.e. in an unambiguous position, typically indicates the presence of a proper name (or of part of a multiword proper name). Although such criterion presents some difficulties, mainly due to case inconsistency³, in this paper we assume that Portuguese names correspond to a word or a string of words

¹Av. Rovisco Pais, 1000 Lisboa, Portugal. E-mail: sme at ist.utl.pt

²Faculdade de Letras da Universidade de Lisboa, Alameda da Universidade, 1600-214 Lisboa, Portugal. E-mail: e_ranchhod at fl.ul.pt

³For instance, *ministro* (minister) can be capitalized or not in the same text: *o ministro*; *o Ministro*.

initialized with a capital letter (e.g. *Lisboa* ‘Lisbon’; *Hospital de Santa Maria* ‘Saint Mary’s Hospital’)⁴.

The challenging problem with candidate named entity strings is the correct delimitation and subsequent classification of entities [2]. In fact, a string of consecutive initial capitalized words – containing potentially functional words, such as determiners and prepositions, or the coordinating conjunction *e* –, can represent either a single name or a series of embedded or independent names, as illustrated by the examples in (1) and (2), respectively:

- (1) a assinatura do *Acordo Geral sobre Tarifas e Comércio*.
(the signing of the *General Agreement on Tariffs and Trade*)
- (2) a vantagem de *Barack Obama sobre Hillary Clinton e John Edwards*.
(the advantage of *Barack Obama over Hillary Clinton and John Edwards*)

Despite their similar structure, example (1) corresponds to a single named entity (the naming of a particular agreement) that contains a preposition (*sobre* ‘on’) and a conjunction (*e* ‘and’), while example (2) illustrates a discourse structure containing three independent person names connected by a preposition and a conjunction. This means that, as with common noun phrases, proper names exhibit structural ambiguity in prepositional phrase attachment and in conjunction scope.

In this paper we are concerned with the disambiguation of proper name strings containing the coordinating conjunction *e*. The ambiguity caused by this conjunction is not negligible. Using simple algorithms, we could estimate that, in a journalistic corpus with 20,463 candidate names, the conjunction occurs 1038 times, which indicates that around 5% of the strings are ambiguous. This proportion is analogous to estimations reported for English before [3], where in a sample of 545 candidate named entity strings, 31 conjunctions were found. Determining the correct analysis of ambiguous strings with coordinating conjunction is important for Portuguese Named Entity Recognition and Classification, and obviously for all applications that rely on named entity extraction.

1. Related Work and Motivation

For Portuguese, it has been observed [7] that the coordinating conjunction *e* may cause structural ambiguity, gave some examples of that ambiguity, and mention that its resolution would require deep syntactic parsing of the text. In the HAREM evaluation contest [8], the proportion of ambiguity and mistagging caused by the erroneous analysis of name coordination have not been evaluated. We participated in the HAREM evaluation, using a system that scored high, but was not capable of handling conjunction in an effective way. The present paper describes the work developed since then on the topic.

For the purpose of this experiment we have considered four semantic classes of proper names: PESSOA (person), names referring to people; LUGAR (location), geographical names referring to countries, cities, mountains, etc., and ORGANIZACAO (organization), names denoting companies, governmental institutions, etc. These are the

⁴Whenever possible and appropriate, we give an approximate translation in English for our Portuguese examples. We omit the translation for person names, and for those names whose spelling coincides largely with that of the original language (e.g. *Bonnie e Clyde* ‘Bonnie and Clyde’).

three main types of names, collectively known as “enamex” since the MUC-6 evaluation (1995). The fourth category, DIVERSO, is a residual, semantically heterogeneous class (“miscellaneous” in the CONLL conferences). It includes naming expressions for events, artifacts, book and movie titles, etc.

2. Linguistic Description

Our concern is the correct delimitation and classification of Portuguese proper names in strings that contain the coordinating conjunction *e*⁵. We have extracted from the corpus all the proper name strings containing at least one and at most two conjunctions. The linguistic analysis of such data let us distinguish two different values of the conjunction: (i) the conjunction is a constituent of the name, i.e. the named entity contains “an internal conjunction” [1], or (ii) the conjunction denotes a relation between two independent names, i.e. the conjunction is name-external. As with common noun phrases [9], when the conjunction links two multi-word independent names some name constituents can be ellipped from one of the names. The ellipses (of part of a name) causes additional difficulty to name identification and classification.

In next sections we describe and illustrate the main linguistic types of coordinated name structures.

2.1. Internal Conjunction

We consider that the conjunction is name-internal if it is an inherent and distinctive element of the proper name. In Portuguese, the coordinating conjunction *e* can be an internal constituent of the four pre-defined semantic types of entities. A few examples of proper names containing conjunctions follow:

- PESSOA: family names (*Maria Brito e Cunha*), artistic groups and bands (*Chutos e Pontapés*; *Despe & Siga*),
- LUGAR: countries (*São Tomé e Príncipe*), streets containing person and geographical names (*Rua Melo e Costa*),
- ORGANIZACAO: institutions (*Câmara de Comércio e Indústria* ‘Chamber of Commerce and Industry’), companies (*Águas do Douro e Paiva*),
- DIVERSO: mentions in texts to events, museums, books, operas, movies etc. (*Crime e Castigo* ‘Crime and Punishment’, *Tristão e Isolda* ‘Tristan and Isolda’, *Bonnie e Clyde* ‘Bonnie and Clyde’).

In all these instances, proper names are multiword nouns, constituted of at least three words, of which one is the lower case coordinating conjunction.

⁵The conjunction *e* can be represented by its variant spelling *&*, but this is rather infrequent in Portuguese. For comparison, in our corpus, there are 20 forms *&*, against 1018 forms *e*. In all these occurrences, *&* is a name-internal conjunction.

2.2. External Conjunction

A totally different and most frequent situation is that where the coordinating conjunction is not part of the name, but it denotes a relation between two independent names. This is the case illustrated by the example:

- (3) O Chile, a Argentina, o Brasil e os Estados Unidos procuram
(Lit. The Chile, the Argentina, the Brazil and the United States seek)⁶

The conjunction appears at the end of an enumeration, linking the last two geographical names.

2.2.1. Ellipsis.

Coordination often involves ellipsis, which is a means of avoiding repetition [9]. For example, the repetition of *Carolina* is avoided in:

- (4) Carolina do Norte e do Sul
(North and South Caroline)

The ellipsis of *Carolina* before *do Sul* (i.e. in the second conjoined noun phrase) leaves the second name lexically incomplete. Following [9], we will call this reduction *anaphoric ellipsis*, since it implies the recuperation of information (a word, in this instance) mentioned before in the discourse structure.

Ellipsis can involve the omission of a word that will be mentioned later in the discourse. For this reason, we will call that reduction *cataphoric ellipsis*. In the following example, the family name *Rocha*, common to *Andrée* and *Clara*, is omitted after *Andrée*, only appearing later, next to *Clara*:

- (5) a mulher e a filha de Torga, *Andrée e Clara Rocha*
(the wife and the daughter of Torga, *Andrée and Clara Rocha*)

These few examples illustrate that the coordination of two proper names often involves the removing of lexical items from one of them: in anaphoric ellipsis, the second name is incomplete; in cataphoric ellipsis, on the contrary, it is the first name that is affected by lexical reduction. In both types of ellipsis the missing words can be exactly recovered (*Carolina do Norte e Carolina do Sul*, example (4); *Andrée Rocha e Clara Rocha*, example (5)).

But in coordinated proper names the ellipted items need not be identical in all respects:

- (6) Câmaras Municipais de Braga e do Porto
(Municipalities of Braga and of Oporto)

Taking into consideration the real world, the interpretation of (6) is obviously that only two municipalities are mentioned in the discourse, i.e. *Câmara Municipal de Braga* and *Câmara Municipal do Porto* (Municipality of Braga and Municipality of Oporto).

⁶In Portuguese, some geographical names may be preceded by articles, that have gender and number inflection: *o, a, os, as* (the).

What is missing in the second conjoined name of (6) is *Câmara Municipal* (and not *Câmaras Municipais*). So, in situations like this one, the correct treatment of ellipsis requires the reconstruction of two noun phrases and the identification of two singular names: *Câmara Municipal de Braga e Câmara Municipal do Porto* (Municipality of Braga and Municipality of Oporto).

The grammars that we have designed can handle ellipsis, but they are not totally satisfactory. This is one of the topics that needs further improvement.

3. Data and Resources

The textual data were extracted from a general-purpose journalistic corpus: *CETEM-Público*. This is a Portuguese untagged public corpus, consisting of excerpts of the Portuguese daily newspaper *Público* that contains about 180 million words (for technical information about the corpus, see [10]).

For the purpose of the work described here, a subcorpus (corpus from now on) with 594,709 tokens, of which 260,071 are words integrated into 11,600 sentences, was drawn randomly from that large corpus.

Using a case-sensitive general tagger, based on a large-scale lexicon, developed previously [11], we identified in that corpus 20,463 candidate named entities, i.e. sequences of at least two initial capitalized words⁷. We obtained 1038 instances of the conjunction: 1018 instances of the conjunction ‘e’ and 20 instances of its variant ‘&’.

We also permit strings to include lower case functional words, articles: *o, a, os, as* (the) and prepositions: *de* (of), *para* (for), *sobre* (on), as well as some punctuation marks (apostrophe, hyphen, comma, quotes). Articles, prepositions, apostrophes and hyphens can be found inside Portuguese multiword names, in particular person and geographical names:

- | | |
|------------------------------|--------------------------------------------|
| (7) África do Sul | (South Africa, lit. ‘Africa of the South’) |
| (8) João d’Ávila | (Person name) |
| (9) Ernesto de Melo e Castro | (Person name) |
| (10) Trás-os-Montes | (Portuguese region) |

In the example (7) the preposition *de* (of) is contracted with an article (*do* = *de+o*). The same preposition is truncated, and an apostrophe replaces the reduced vowel, in example (8). The aristocratic, or aristocratic-like, family name of example (9), *de Melo e Castro*, incorporates the preposition *de* and the conjunction *e*. In (10), *Trás-os-Montes*, as well some other geographical (e.g. the city *Dar-es-Salam*) and organization names, contains an internal hyphen linking the lexical constituents. A few complex names (i.e. multiword names with nested names) can contain commas:

- (11) Seminário de História Medieval, Moderna e Contemporânea
(Seminar in Medieval, Modern and Contemporary History)

In general, commas are found in independent name apposition, and, since the conjunction represents the end of an enumeration, it can be used as an external evidence for

⁷In the present study, we have restricted ourselves to candidate named entity strings that contain a single conjunction.

conjunction analysis. Brackets and quotes include named entities but they are not found inside names.

We did not consider as candidate name entities capitalized words appearing after a punctuation mark requiring capitalization, since most of them correspond to single (i.e. not conjoin) uppercase initial words, the majority of which are articles and other functional words.

For parsing and tagging proper names, in addition to the general tagger, we used the following lexical, syntactic and semantic resources: a gazetteer comprising around 16,000 lexical entries in total, a list of 337 designators and a list of 380 trigger words indicating or accompanying persons, organizations, locations, etc.; a set of finite-state grammars (70), which use the lexical knowledge represented in the gazetteers, designator and trigger word lists, and describe relevant local syntactic information.

The experiments were conducted using Unitex, a modular open source toolkit based on finite-state technology [13].

In next paragraphs we describe in more details these data and resources.

3.1. Gazetteers.

Lexical entries are single and multiword person, location, organization and other proper names. Some names with internal conjunction are included in these word lists (e.g. the Portuguese surname *Melo e Castro* (Cf. example 10)). Lists of names used contain:

person: about 6,500 person names; *location*: about 5,500 major country, province, state, city and town names; *organization*: about 3,000 company and governmental and public institution names; *diverse*: about 1000 proper names mentioning gardens, bridges, months, books, newspapers, etc.

The word lists have been extracted semi-automatically from texts and word lists published on the Web, and enriched manually with linguistic features afterwards [11]. Linguistic attributes include PoS, usually Noun (N), morphological (when appropriate) and semantic classification. For example, *N+Geo+Gep* means that the entry is a noun (N), a geographical name (Geo) of subtype geo-political (Gep).

3.2. Designator and trigger word lists.

We distinguished between designators and trigger words. Designators are lower case words found in the left or right context of a pre-defined pattern of initial capitalized words that permit to include them in a class, even subclass, of proper names. For example, *provincia de* (province of) and *rio* (river) may consistently allow geographical names to be determined and included in the semantic class LUGAR (location): *provincia de Cabinda*; *rio Ganges*. Nouns indicating human occupations (e.g. *arquitecto-arquitecta* ('architect', masculine and feminine), *professor-professora* (professor) often precede person names (professor *Moniz Pereira*). Designators are particularly useful to recognize and classify foreign proper names not included in the gazetteers, but mentioned in journalistic texts (e.g. professor *Donald Kettl*; actriz (actress) *Helen Mirren*; armazéns (store) *Marks and Spencer*).

Trigger words, in turn, are upper cased words that are included in multiword names and permit, as designators do, the class of that names to be discovered. For example, human titles (e.g. Sr., Eng., Prof., Ministro) are regularly upper cased words and ab-

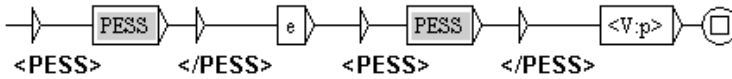


Figure 1. A graphical representation of a finite-state transducer that implements the rule $P1 \text{ e } P2$.

abbreviations accompanying person names; *Companhia de Seguros* (Assurance Company), *Grupo Segurador* (Assurance Group) and *Instituto* (Institute) are examples of trigger words included in organization names (companies and institutions, respectively).

We took advantage of morphological features of Portuguese (in particular, singular, plural) and have listed separately singular and plural designator and trigger words. A designator in the plural before a string that contains two conjoined proper names is in general an external evidence for two independent names: *tenores José Carreras e Plácido Domingo* (tenors José Carreras and Plácido Domingo); *ilhas Terceira e São Miguel* (islands Terceira and São Miguel), *rios Ardila e Guadiana* (rivers Ardila and Guadiana).

Designators and trigger words were manually collected by applying the dictionary of proper names to texts, and looking at their context in those texts. Using Unitex facilities, designators and trigger words were compiled into a finite-state recognizer to build and tag proper names.

3.3. Finite-state local grammars.

Designators and trigger words are very helpful for finding names and for classifying them semantically, but they are insufficient to handle accurately the conjunction's ambiguity.

Based on linguistic analysis (see sections 3.1, 3.2.), we have written grammar rules to handle the ambiguity. Such rules describe restricted patterns and constraints specific to each class of coordinated proper names. There are 70 rules in total, of which 23 for person, 15 for organization, 9 for location, and 23 for the semantically heterogeneous group. The set of these context-dependent rules can be viewed as a *local grammar* [13] for proper names. Local grammars capture typical patterns associated to designators and trigger words, but they can also represent accurately dependencies between words, and general syntactic relations (e.g. subject-verb relations), without applying to more powerful syntactic formalisms. They also make use of part of speech tags as well as of linguistic information included in the gazetteers.

Local grammars can be efficiently compiled into finite state transducers [13], and then be applied to texts to parse proper names. We compiled our grammars into FST using Unitex tools.

Fig. 1 represents a finite-state transducer that implements a local rule (or a local grammar [14]) that captures two independent coordinated person names.

The rule $P1 \text{ e } P2$ recognizes two coordinated person names that are the subject of a plural verb form ($<V:p>$ ⁸). The rule assumes that, in such conditions, the conjunction is name-external. The rule is represented in a graph that references a person name sub-grammar, *PESS*, i.e. the main graph calls the sub-graph *PESS*. This sub-graph is partially represented in Fig. 2.

The sub-grammar *PESS* recognizes as a person name any noun encoded in the gazetteers as $<N+Pes>$, which may be preceded by an adequate trigger word (TW_{P1})

⁸Notice that the corpus has been previously tagged by a case-sensitive general tagger based on a comprehensive lexicon.

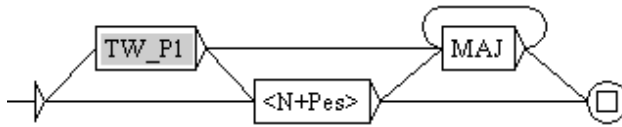


Figure 2. A finite automaton representing the sub-grammar PESS.

and followed by any string of uppercase words (*MAJ*); it also predicts that any string of initial capitalized words preceded by a person trigger word is a person name.

In Fig. 1, the transducer outputs represent the sequences of tags that will be inserted into the text. The results of running the rule *P1* e *P2* over the corpus are illustrated by the examples:

- (12) <PESS>Pinto Balsemão/<PESS> e <PESS> D. José Policarpo</PESS>
foram felicitados
- (13) <PESS>João Paulo II</PESS> e <PESS>Fidel Castro</PESS>
trataram, com a formalidade
- (14) <PESS>Cavaco Silva</PESS> e <PESS>António Guterres</PESS>
conseguiram, cada qual a

The collection of the seventy local grammars has been combined and compiled into a single finite-state transducer grammar, which was applied in one pass of the parser to proper name processing.

4. Results and Evaluation

After the processing of the corpus, using Unitex and the linguistic knowledge described in 4., the system produced a tagged version of the original text. In this annotated corpus all the identified proper names, appearing in coordinated strings, were marked up with SGML tags that specify their semantic class. For reminding, we have constituted four semantic classes, tagged as: <PESS> for person names, <ORG> for organizations, <LUG> for locations, and <DIV> for other diverse semantic types of naming expressions.

The annotation processing also took into account the conjunction type present in the string (name-internal, name-external, elliptical construction). The tagged corpus was automatically scored against the manual tagging of the same text performed by a linguist, who identified 1023 proper names involving the conjunction (*e* and *&*). The overall results of that evaluation are shown in Table 1.

Table 1 shows that, in our journalistic corpus, the more significant set is constituted of independent proper names linked by the conjunction (80%). Of these only a small proportion (1,5%) is affected by ellipsis. The contribution of the system to the analysis of the nature of the conjunction is also illustrated. The system scored satisfactorily, except for cataphoric ellipsis. However, the small number of instances of such constructions makes the results statistically insignificant. On the other hand, ellipsis detection may be a non trivial problem even for a human expert. Yet, these results indicate that more sophisticated resources and more extensive data are needed for analyzing elliptic constructions in an adequate manner. The best results are for name-internal conjunction.

Table 1. Overall Precision and Recall Scores

	Name Internal Conjunction	Name External Conjunction			Total
		With no Ellipsis	Anaphoric Ellipsis	Cataphoric Ellipsis	
Manual	205	803	11	4	1023
Tags	180	720	9	1	910
Correct	172	607	9	1	789
Precision	0.96	0.84	1.00	1.00	0.86
Recall	0.84	0.76	0.82	0.25	0.77
F-measure	0.89	0.80	0.90	0.40	0.82

Table 2. Overall Scores of Semantic Analysis

Semantic Class	Manual Tagging	System annotation				
		Tags	Correct	Precision	Recall	F-measure
PESS	727	635	589	0.93	0.81	0.86
ORG	443	468	350	0.75	0.79	0.77
LUG	512	411	382	0.93	0.75	0.83
DIV	164	127	108	0.85	0.67	0.74
Total	1846	1641	1429	0.87	0.77	0.82

Table 3. Name-internal Conjunction Scores

Semantic Class	Manual Tagging	System annotation				
		Tags	Correct	Precision	Recall	F-measure
PESS	45	39	39	1.00	0.87	0.93
ORG	99	94	88	0.94	0.89	0.91
LUG	16	9	8	0.89	0.50	0.64
DIV	45	38	37	0.97	0.82	0.89
Total	205	180	172	0.96	0.84	0.89

The human annotator also included proper names into one of the four pre-defined semantic classes. Table 2 shows the human analysis, and evaluates the results not only for the system as a whole, but for each semantic class of names.

As a complementary view to conjunction analysis, we have examined in more details the data involving name-internal conjunction. Table 3 shows the achieved scores for each semantic class of proper names.

The F-measure is better than its value in Table 2. The fact that the conjunction variant & (20 occurrences) is always internal to the name may contribute to this result.

5. Conclusion

In this work we have concentrated on the problem of the correct delimitation and classification of Portuguese proper names in strings that contain the coordinating conjunction e

(and). Such strings are ambiguous, since the conjunction is either name-internal, i.e. it is part of the proper name, or name-external, i.e. it denotes a relation between two names.

We have shown that the various types of ambiguity can be handled satisfactorily using a small semantic lexicon and finite-state local grammars. Such local grammars do not just capture typical patterns associated to proper names, but they can also describe accurately dependencies between words, and local syntactic relations.

We have evaluated quantitatively our results against a manual tagging performed by a linguist. Unsurprisingly, the results confirm that linguistic knowledge based rules obtain better precision scores in comparison with the recall values.

In order to improve the work reported here, we believe it is necessary to further develop the syntactic component, by incorporating new and more precise local grammars.

References

- [1] D. Maurel, Les mots inconnus sont-ils des noms propres?, in: Actes des journées internationales d'analyse statistique des données textuelles, Presses Universitaires de Louvain, Louvain (2004).
- [2] S. Galicia-Haro and A. Gelbukh, Complex named entities in Spanish texts: Structures and properties, in: S. Sekine and E. Ranchhod (Eds.), Named Entities: Recognition, classification and use, special issue of *Linguisticae Investigationes*, **30:1** (2007), 69-94.
- [3] P. Mazur and R. Dale, Handling Conjunctions in Named Entities, in: S. Sekine and E. Ranchhod (Eds.), Named Entities: Recognition, classification and use, special issue of *Linguisticae Investigationes*, **30:1** (2007), 49-68.
- [4] D. Nadeau and S. Sekine, A survey of named entity recognition and classification, in: S. Sekine and E. Ranchhod (Eds.), Named Entities: Recognition, classification and use, special issue of *Linguisticae Investigationes*, **30:1** (2007), 3-26.
- [5] L. Rau, Extracting Company Names from Text, in: Proceedings of the Seventh Conference on Artificial Intelligence Applications of IEEE (1991).
- [6] D. McDonald, Internal and External Evidence in the Identification and Semantic Categorization of Proper Names, in: Proceedings of SIGLEX Workshop on Acquisition of Lexical Knowledge from Text (1993).
- [7] C. Mota, D. Santos and E. Ranchhod, Avaliação de reconhecimento de entidades mencionadas: princípio de AREM, in: D. Santos (Ed.), *Avaliação conjunta: um novo paradigma no processamento computacional da língua portuguesa*, IST-Press, Lisboa, 2007.
- [8] D. Santos, N. Seco, N. Cardoso and R. Vilela, HAREM: an Advanced NER Evaluation Contest for Portuguese, in: Proceedings of LREC (2006).
- [9] R. Quirk, S. Greenbaum, S. Leech and J. Svartvik, *A Grammar of Contemporary English*, Longman Group, Ltd., 1980.
- [10] D. Santos and P. Rocha, Evaluating CETEMPúblico, a free resource for Portuguese, in: Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics (2001).
- [11] E. Ranchhod, C. Mota and P. Carvalho, Portuguese Large-scale Language Resources for NLP Applications, in: Proceedings of the IV Conference on Language Resources and Evaluation, LREC (2004).
- [12] S. Paumier, Unitex 1.2. User Manual, in: <http://www-igm.univ-mlv.fr/~unitex/>
- [13] M. Gross, The Construction of Local Grammars, in: E. Roche and Y. Schabes (Eds.), *Finite-State Language Processing*, The MIT Press, Cambridge, Massachusetts, 1997.
- [14] M. Mohri, Local Grammar Algorithms, in: Inquiries into Words, Constraints, and Contexts. Festschrift in Honour of Kimmo Koskenniemi on his 60th Birthday. CSLI Publications (2005).

A Memory-efficient ε -Removal Algorithm for Weighted Acyclic Finite-State Automata

Thomas HANNEFORTH

Institut für Linguistik, Universität Potsdam

Abstract. Many NLP tasks based on finite-state automata create acyclic result automata which contain a lot of ε -transitions. We propose an refinement of an existing algorithm for ε -removal with a better memory consumption behavior in many practical cases.

Keywords. Weighted finite-state automata, semirings, computational linguistics

Introduction

Many NLP tasks based on finite-state automata (FSA) create results which contain a lot of ε -transitions. In most of the cases, the results are acyclic since the input is usually a finite language like a set of sentences. The ε -transitions have to be removed due to speed and memory efficiency reasons. Two examples should suffice to illustrate the problem: N -gram counting [1] and weighted local grammar application [2,3].

Fig. 1 shows a toy corpus C as a weighted finite-state automaton (WFSA) over the real semiring (cf. section 1.1), while fig. 2 exemplifies a bigram counter $Counter^2$. The

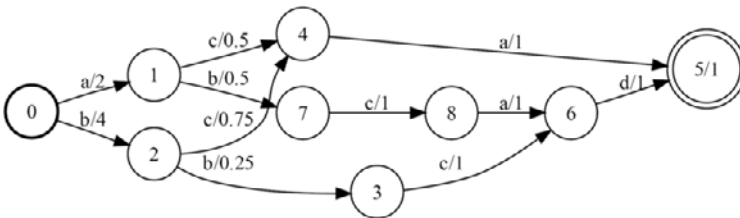


Figure 1. Corpus C as WFSA (the weights along a path labeled with x must be multiplied to compute the number of occurrences of x in the corpus)

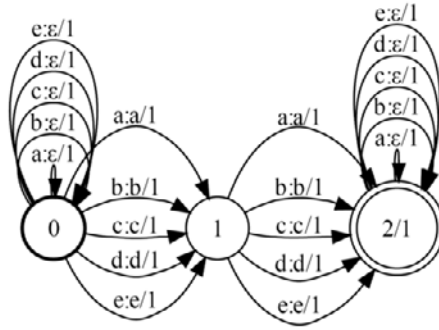


Figure 2. Bigram counter $Counter^2$

composition of the two, followed by taking the 2^{nd} projection (that is, taking the lower tape) results in the weighted FSA shown in fig. 3¹.

Due to the non-determinism in the bigram counter (the transducer in fig. 2 can choose in state 0 whether to stay in the loop and map prefixes of the input to ε or go to state 1 and start counting a specific bigram), the result contains ε -subgraphs at the beginning and at the end of each bigram counted. Since these ε -transitions prevent further optimisations like determinisation and minimisation, their removal is an important prerequisite of these further processing steps.

The second example is from the field of local grammar application. A local grammar consists of a collection of "interesting patterns" (for example noun phrase or named entity patterns) which are applied to some input text. In many cases the part which is matched by some rule in the local grammar is surrounded by brackets and the non-matched parts are mapped to ε . Consider a rule²

$$a(b^+) \rightarrow^\varepsilon [\dots] . \quad (1)$$

Applied to an input $I = acabbbac$ (again by composition followed by 2^{nd} projection and mapping all material outside the brackets to ε) this leads to the FSA shown in fig. 4.

Again, we find long ε -chains in the resulting FSA.

¹The path 0 1 16 14 8 7 in fig. 3 accounts for the bigram bc found along the path 0 1 7 8 6 5 in fig. 1. The bigram counter has chosen to map the a prefix and the ad suffix along this path to ε .

²The operator \dots is a symbolic placeholder for the pattern on the left hand side of the rule to be circumfixed with the brackets (cf. [4]). The operator \rightarrow^ε means that input which remains unbracketed is mapped to ε .

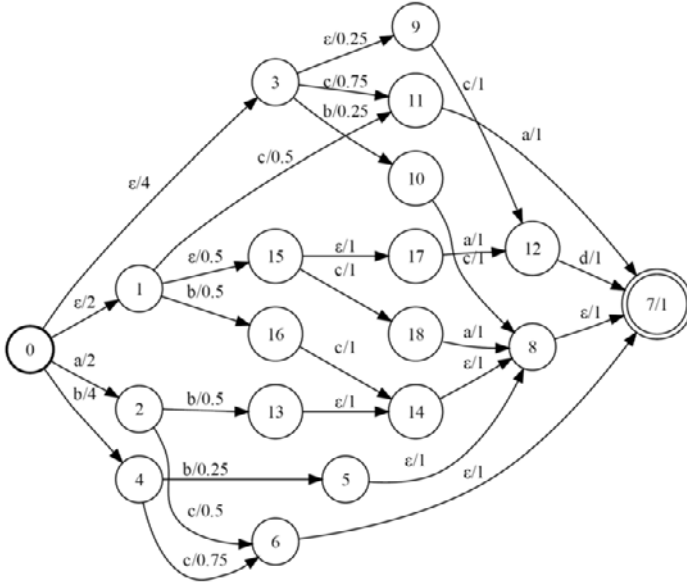


Figure 3. The counted bigrams $\Pi_2(C \circ \text{Counter}^2)$

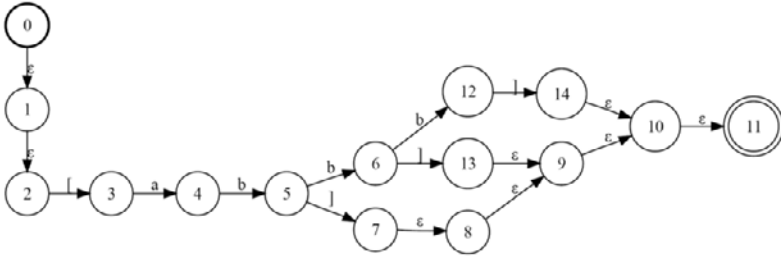


Figure 4. Result of the application of rule (1) to *acabbbaac*

Mohri's ε -removal algorithm

Mohri's algorithm ([5], which is a generalisation of an algorithm presented in [6]:76 to the weighted case) works in three steps, which are shown in table 1.

The mode of operation of the algorithm is depicted in fig. 5.

Mohri also discusses the special case of ε -removal if the ε -subgraph is acyclic. In that case, the first two steps of the algorithm are actually interleaved: the states of the ε -subgraph are processed in reverse topological order and the ε -paths of step 1 are reduced

1. For each state p compute the ε -distance to any other reachable state q . Then delete all ε -transitions from the WFSA.
2. For each state pair p and q with ε -distance w and a single transition from q to r labeled with $a \in \Sigma$ and weight w' , add a transition from p to r with label a and weight $w \otimes w'$ to the WFSA. If q is a final state, p will also become a final state. If p already was a final state, the final weights of q and p are additively combined.
3. Remove all non-reachable states and all their ingoing and outgoing transitions.

Table 1. Mohri's ε -removal algorithm (cf. section 1 for notation)

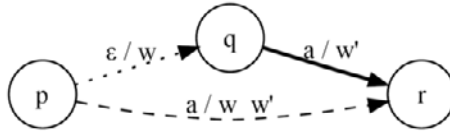


Figure 5. The core of Mohri's ε -removal algorithm: whenever the ε -distance between states p and q is w and a there is a single transition from q to r labeled with a and having weight w' , add a new a -transition from p to r with combined weight $w \otimes w'$. Note that p and q or q and r may denote the same state, even $p = q = r$ is possible.

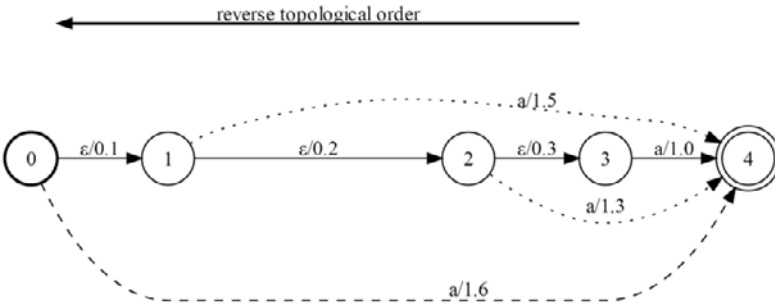


Figure 6. ε -removal by processing the states in reverse topological order (weights are added here).

to single ε -transitions. Consider the example in fig. 6. The states are processed in the order 3, 2, 1, 0. At each state p the ε -removal pattern of fig. 5 is applied and the a -transitions, which accumulate the weight of the ε -transitions are “pulled” towards the start state.

Both versions of the algorithm share an efficiency problem which also can be seen in fig. 6: the algorithm attaches a lot of transitions to states (like 1 and 2) which become unreachable after step 1 and are removed together with their outgoing transitions in step 3. Only the dashed transition from state 0 to state 4 remains. In tasks like N -gram

counting for big corpora and local grammar application for long input texts, this leads to a growth in memory consumption which may prevent the successful application of the algorithm. In section 2, we report some experiments in which we compare different versions of ε -removal algorithms. Before that, section 1 will introduce an improvement of the algorithm.

1. An improved algorithm

The idea to avoid adding transitions which do not contribute to the resulting FSA is quite simple: add a data structure which registers the reachable states after step 1 of the algorithm in table 1 and maintain it during step 2. Before we present the revised version of the algorithm, some definitions and notational conventions are required. Let $A = \langle \Sigma, Q, q_0, F, E, \rho \rangle$ be a weighted finite-state acceptor with alphabet Σ , state set Q , start state q_0 , and set of final states F . The set of transitions E is a subset of $Q \times \Sigma \times \mathcal{W} \times Q$. \mathcal{W} is a set of weights being the carrier set of a semiring (see below). Given a transition $t \in E$ we denote with $w[t] \in \mathcal{W}$ the weight associated with t , with $n[t]$ the destination state of t and with $l[t] \in \Sigma$ the label of t . We make use of an adjacency list representation, so $E[p]$ denotes the outgoing transitions of state p . The final weight function ρ is a function mapping final states to weights.

A *path* between states p and q is a sequence of adjacent transitions leading from p to q . In that case, we call q *reachable* from p . An ε -*path* is a path between p and q where the concatenation of the labels along the path yields ε . In that case, we call q ε -*reachable* from p . If a path between p and q is labeled with $x \neq \varepsilon$, we say that q is Σ -*reachable* from p . A state p is *reachable* when it is reachable from the start state q_0 .

The revised version of the ε -removal algorithm, restricted to acyclic WFSAs, can be found in table 2. The function `collect-weights(A)` in line 1 combines transitions with similar source state, destination state and label by combining their weights (with abstract addition). This step is not strictly necessary but potentially reduces the number of transitions to consider. Since reachability means *start state reachability*, the algorithm operates in normal (forward) topological sort order. The topological sort in line 2 ensures that for every transition $p \rightarrow q$ the state number of q is strictly greater than the state number of p . The acyclicity of the input FSA A guarantees that such an ordering is always possible.

In line 3 we initialise the set of reachable states R with the reflexive and transitive closure of the set q_0 under the Σ -*reachability* relation. After that, R contains all states which are reachable by regular symbols starting at q_0 .

Lines 4 to 23 constitute the main loop of the algorithm. All states $p \in Q$ which have outgoing ε -transitions are processed in ascending (that is topological) order. If $p \in R$ we apply the ε -removal pattern (cf. fig 5) to it: for each ε -reachable state p determine the set of pairs $\langle q, w \rangle$, w being the ε -distance from p to q . This task is delegated to a function `compute-shortest- ε -distances` and can be implemented in different ways (see below). Then find a -transitions t (with $a \in \Sigma$) to states r in q 's adjacency list $E[q]$ and connect p and r with a transition t' with $l[t'] = l[t]$ and $w[t'] = w \otimes w[t]$. Since p was reachable, the same is now true for r , so we add r to a set R' . After that, the closure of R' under the reachability relation is computed and the resulting set is merged with R .

After the main loop, the function `delete- ε -transitions` in line 24 deletes all ε -transitions from A . Finally, line 25 deletes all non-reachable states $Q - R$ from A .

Require: An acyclic WFSA $A = \langle \Sigma, Q, q_0, F, E, \rho \rangle$
Ensure: An equivalent ε -free WFSA A'

```

1: collect-weights( $A$ )
2: topsort( $A$ )
3:  $R \leftarrow \Sigma$ -reachable( $\{q_0\}$ )
4: for all  $p \in Q$  in ascending order do
5:   if  $p \in R$  then
6:      $d \leftarrow \text{compute-shortest-}\varepsilon\text{-distances}(A, p)$ 
7:      $R' \leftarrow \emptyset$ 
8:     for all  $q$  with  $d[q] \neq \bar{0}$  do
9:       for all  $t \in E[q]$  with  $l[t] \neq \varepsilon$  do
10:         $E \leftarrow E \cup \langle p, l[t], d[q] \otimes w[t], n[t] \rangle$ 
11:         $R' \leftarrow R' \cup n[t]$ 
12:      end for
13:      if  $q \in F$  then
14:        if  $p \in F$  then
15:           $\rho(p) \leftarrow \rho(p) \oplus (d[q] \otimes \rho(q))$ 
16:        else
17:           $F \leftarrow F \cup \{p\}$ 
18:           $\rho(p) \leftarrow d[q] \otimes \rho(q)$ 
19:        end if
20:      end if
21:    end for
22:     $R \leftarrow R \cup \Sigma$ -reachable( $R'$ )
23:  end if
24: end for
25: delete- $\varepsilon$ -transitions( $A$ )
26: connect( $A$ )
27: return  $A$ 

```

Table 2. ε -removal algorithm with reachability constraint

1.1. Computing ε -distances

The ε -removal algorithm in table 2 relies on an auxiliary function which computes the ε -distances. Before we come to that we need a suitable weight structure to quantify these distances.

An algebraic structure $\langle W, \oplus, \otimes, \bar{0}, \bar{1} \rangle$ is a semiring [7] if it fulfills the following conditions:

1. $\langle W, \oplus, \bar{0} \rangle$ is a commutative monoid with $\bar{0}$ as the identity element for \oplus .
2. $\langle W, \otimes, \bar{1} \rangle$ is a monoid with $\bar{1}$ as the identity element for \otimes .
3. \otimes distributes over \oplus .
4. $\bar{0}$ is an annihilator for \otimes : $\forall w \in W, w \otimes \bar{0} \otimes w = \bar{0}$.

A common instantiation of a semiring is the *real semiring* $K_{\mathbb{R}^+} = \langle \mathbb{R}^+, +, \cdot, 0, 1 \rangle$ where abstract addition and multiplication coincide with their non-abstract counterparts.

A path π is defined as a sequence of adjacent transitions. The weight $w[\pi]$ of a path $\pi = t_1 t_2 \cdots t_k$ is defined by $w[t_1] \otimes w[t_2] \otimes \cdots \otimes w[t_k]$. Let $\Pi(P, x, Q)$ be the set of all paths with source state $p \in P$, destination state $q \in Q$ and labeled with $x \in \Sigma^*$. The ε -distance between p and q is defined as follows:

$$\varepsilon\text{-dist}(p, q) = \bigoplus_{\pi \in \Pi(\{p\}, \varepsilon, \{q\})} w[\pi] \quad (2)$$

Thus, the ε -distance between states p and q might consist of several ε -paths whose weights are additively combined.

Define a ε -subautomaton of A as a maximal (wrt ε -reachability) connected subautomaton of A with transitions only labeled with ε . While processing the input FSA A in topological order we find a number of these ε -subautomata, since every state p not being ε -reachable and having outgoing ε -transitions constitutes the start state of a subautomaton. If the states of a ε -subautomaton rooted in p are also processed in topological order, it is guaranteed by the *path-relaxation property* (cf. [8]:609f) that `compute-shortest- ε -distances(A, p)` correctly computes the ε -distances for all states $q \in Q$ according to equation (2).

But the topologically ordered state subsequences of two ε -subautomata of A may be interleaved with respect to the topologically ordered state sequence of A itself. Furthermore, a state can be part of more than a single ε -subautomaton. Consider the (already topologically ordered) WSFA in fig. 7. There are two ε -subautomata, one rooted at state 1 with topologically ordered state sequence 1 3 4 5 and another rooted at state 2 with sequence 2 4 5.

So, the problem is, how to compute the ε -distances efficiently, preferably by making use of the already topologically ordered state sequence of the complete automaton. We are aware of four solutions for that:

1. Use the topological order of the complete automaton to compute the ε -distances in the ε -subautomata, eventually skipping states which are not part of the ε -subautomaton currently under investigation.
2. Start a depth-first search from every start state of a ε -subautomaton, thereby constructing a topological order for the states in the subautomaton, which is processed in a second pass to compute the distances.
3. Variant of 2: Cache already computed ε -distances thus avoiding recomputation.
4. Use a special queue discipline to make use of the topological sort order computed in line 2 in table 2.

The last item 4. in the list above needs some elaboration. Since line 2 of the algorithm in table 2 guarantees that the destination state of a transition is strictly greater than its source state, we put the states into a priority queue ordered by state number. The ordering of the state priority queue by state number ensures that all states in the ε -subautomaton are processed in topological order. The test in line 6 avoids processing any state in the subautomaton twice.

1.2. Correctness and complexity

Since the states of the WSFA A are visited in topological order – the same is true for the states in the ε -subautomaton – all ε -distances are correctly computed. A state q for which the test in line 6 fails – that is, q is not reachable from the start state q_0 – cannot become reachable later during the execution of the algorithm, since that would imply the presence of *back edges* in the sense of [8]:546, which would entail a cyclic WSFA, contrary to our assumption. To summarise, the algorithm doesn't miss paths contributing to the weighted language of the WSFA. For the language contributing paths, the proof in [5]:Theorem 1 carries over.

The worst case time complexity of the algorithm is $|Q|$ times the complexity of the ε -distance computation step. This worst case would be a WSFA where for every

Require: A WFSA $A = \langle \Sigma, Q, q_0, F, E, \rho \rangle$ and a source state p

Ensure: A vector d ; $d[s]$ contains for each state s reachable from p the ε -distance between the two

```

1:  $S \leftarrow \emptyset$ 
2:  $PQ \leftarrow \emptyset$ 
3: enqueue( $PQ, p$ )
4: while  $PQ \neq \emptyset$  do
5:    $q \leftarrow \text{dequeue}(PQ)$ 
6:   if  $q \notin S$  then
7:      $S \leftarrow S \cup \{q\}$ 
8:     if  $q = p$  then
9:        $d_q \leftarrow \bar{1}$ 
10:    else
11:       $d_q \leftarrow d[q]$ 
12:    end if
13:    for all  $t \in E[q]$  with  $l[t] = \varepsilon$  do
14:       $d[n[t]] \leftarrow d[n[t]] \oplus (d_q \otimes w[t])$ 
15:      enqueue( $PQ, n[t]$ )
16:    end for
17:  end if
18: end while
19: return  $d$ 

```

Table 3. compute-shortest- ε -distances(A, p)

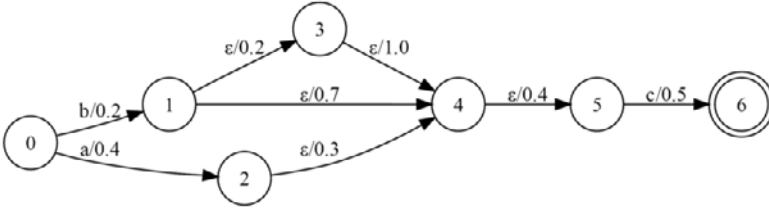


Figure 7. A WFSA with two ε -subautomata

state p all of p 's successor states in topological order were reachable by sequences of alphabet symbols as well as by ε -paths. The complexity of the ε -distance computation is in $O(|Q| + |E|)$ if the topological order method (2. in the enumeration above) is used. The strategy which uses a priority queue is in the complexity class $O((|Q| \log |Q|) + |E|)$.

2. Experiments

We implemented the algorithm with in the FSM<2.0> framework, a C++ template library which allows to use weighted finite-state machines with predefined or user-defined semirings [9]. To compare different strategies for the computation of ε -distances, we implemented methods 2. and 4. of the list in subsection 1.1 above. We compared these two methods with an implementation of the algorithm proposed for acyclic FSAs in [5] and exemplarily displayed in fig. 6. All algorithms were coded very carefully. We didn't

	Total time (in s)	Max. memory usage (in MB)	# transitions (before connect)
1. processing in reverse topological order (cf. fig. 6)	3.48	409	13,306,056
2. processing the ε -subautomata in topological order	8.46	116	2,912,740
3. using a priority queue (cf. table 3)	8.21	106	2,912,740

Table 4. Using the ε -removal algorithm with different ε -distance computation strategies

further investigate method 1. and 3. in the enumeration in section 1.1 since preliminary tests with smaller input FSAs indicated that the caching strategy consumed an unreasonable amount of memory, while the strategy, which skips states not part of the currently processed ε -subautomaton needed too much time.

The point of origin for the experiments were the sentences of the Tiger treebank [10], a German newspaper treebank with around 50.000 trees. Its sentences were compiled into an optimised WFSA over the real semiring with 681,689 states and 730,175 transitions. To this WFSA, a trigram counter similar to the one shown in fig. 2 was applied, resulting in a WFSA with 2,724,212 states and 3,615,890 transitions, among the latter were 1,429,530 ε -transitions. The out-degree, that is, the maximum number of outgoing transitions for a state was 14,044, the size of the alphabet of both WFSAs was 89,418.

The following table shows the processing time, memory consumption and number of transitions before the final connection step³ for the three ε -distance computation strategies described above. Of course, the WFSA after ε -removal was the same in all cases: it contained 2,045,059 states (681,686 of these final) and 2,185,500 transitions.

Not surprisingly, the acyclic version of Mohri's algorithm is very fast⁴, since it only requires a single traversal through the state sequence. But, 83.5 % of the added transitions were useless, thus leading to the observed increase of memory usage. For input automata with a size of the next order of magnitude, the application of this version of the algorithm may become unfeasible. This will especially become true for corpus processing tasks like N -gram counting, since larger corpora entail bigger alphabet sizes, which in turn lead to bigger out-degrees of states like q in fig. 5. If all these transitions are attached to states which may be subject to deletion in the final connection step, superfluous, time and space consuming computations are a matter of fact.

The two instantiations of the ε -removal algorithm with enforced reachability constraint behave almost equally. The priority queue version of table 4 is slightly better, although it uses a queue with $n \log n$ complexity. In both cases, only 25 % of the transitions must have been removed. This is unavoidable since there always may exist states whose outgoing transitions become useless after deleting the ε -transitions leading to them (for example, state 3 in fig. 6).

³All experiments were run on a system with Intel QuadCore 2.66 GHz CPU and 4 GB RAM. We used `float` for the weights of the WFSa and `int` for states and symbols.

⁴Our implementation of the algorithm benefits significantly from the very fast underlying FSA representation in the utilised software library, which is capable of adding/deleting over 10 million transitions per second.

3. Conclusion

We presented a memory efficient algorithm for removal of ε -transitions in acyclic, weighted automata and conducted some experiments.

Acknowledgments I would like to thank Kay-Michael Würzner for very helpful comments on an earlier version of this article.

References

- [1] C. Allauzen, M. Mohri, and B. Roark. Generalized Algorithms for Constructing Statistical Language Models. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, volume 41, pages 40–47. The Association for Computational Linguistics, 2003.
- [2] M. Gross. *The construction of local grammars*. MIT Press, Cambridge, 1997.
- [3] M. Mohri. *Local grammar algorithms*. CSLI Publications, Stanford, 2005.
- [4] K. R. Beesley and L. Karttunen. *Finite State Morphology*. CSLI, 2003.
- [5] M. Mohri. *Generic epsilon-removal algorithm for weighted automata*, volume 2088 of *Lecture Notes in Computer Science*. Springer, Heidelberg, 2001.
- [6] S. Sippu and E. Soisalon-Soininen. *Parsing Theory*, volume I: Languages and Parsing. Springer, 1988.
- [7] W. Kuich and A. Salomaa. *Semirings, Automata, Languages*, volume 5 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1986.
- [8] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 2nd edition, 2001.
- [9] T. Hanneforth. FSM<2.0> – C++ library for manipulating (weighted) finite automata. <http://www.ling.uni-potsdam.de/~tom/fsm>, 2004.
- [10] S. Brants, S. Dipper, S. Hansen, W. Lezius, and G. Smith. The TIGER treebank. In *Proceedings of the Workshop on Treebanks and Linguistic Theories*, Sozopol, 2002.

Regular Expressions and Predicate Logic in Finite-State Language Processing

Mans HULDEN
University of Arizona

Abstract. This paper proposes an extension to the formalism of regular expressions with a form of predicate logic where quantified propositions apply to substrings. The implementation hinges crucially on the manipulation of auxiliary symbols which has been a common, though previously unsystematized practice in finite-state language processing. We also apply the notation to give alternate compilation methods for two-level grammars and various types of replacement rules found in the literature, and show that, under a certain interpretation, two-level rules and many types of replacement rules are equivalent.

Introduction

The popularity of finite-state natural language processing can probably be partly attributed to the expansion of the the idiom of regular expressions—that is, the introduction of new regular expression operators that provide increasing layers of abstraction upon simpler regular expressions in automaton and transducer construction. Instead of designing finite-state automata or transducers manually, or even using a basic set of regular expression operators, the finite-state developer now has an array of flexible operations to choose from, including two-level rules, various flavors of rewrite rules, directional longest-match rules, context restriction rules, priority union, lenient composition, etc [1,2].

Many of these operators are naturally defined in terms of simpler regular expressions—for instance, many finite-state toolkits have a separate operator with the semantics of “contains exactly one instance of a substring drawn from the language \mathcal{L} ,” \mathcal{L} being an arbitrary regular language. The same idea can of course be expressed through the more cumbersome and less legible:

$$(\Sigma^* \mathcal{L} \Sigma^*) - (\Sigma^* ((\Sigma^+ \mathcal{L} \Sigma^* \cap \mathcal{L} \Sigma^*) \cup (\mathcal{L} \Sigma^+ \cap \mathcal{L})) \Sigma^*) \quad (1)$$

Unlike this example, a good many of the more advanced expressions—such as contextual rewriting, or directed replacement—are very difficult to define through basic regular expressions. The common solution throughout the research and implementation of these advanced regular expression operators has been the use of “auxiliary marker symbols”—symbols that in the process of compiling complex statements are inserted into a language, constrained, and finally removed.

For instance:

- [3] use “auxiliary brackets” to develop a rule compiler for two-level rules, where a significant portion of the description of the method is devoted to complications in compiling “overlapping” restriction rules.
- [4] make extensive use of “auxiliary brackets” which are inserted, whose presence is constrained, and which are then appropriately ignored in some contexts in defining rewrite rules and two-level rules as regular relations.
- [5], [6], and [7] use various bracketing systems to define replacement, directed replacement, and parallel replacement rules.
- [8] define a context restriction operator (\Rightarrow), as well as a more generalized context restriction operator, through the use of a \diamond -symbol, whose occurrence is constrained and which is then removed.

The possible drawbacks with this method, though very expressive, include the difficulty of post-design analysis of complex constructions, as well as verification of their correctness. In light of these problems, it would be desirable to at least systematize the notation, if not to abstract the entire construction method under a new notation where the component parts would be easy to verify for semantic correctness, and which would also be easily extended to allow the description and finite-state compilation of novel expressions.

This paper presents such an abstraction of the technique of auxiliary symbol usage in designing complex regular languages and relations. We find that, if defined in an appropriate manner, a particular kind of abstraction of this technique is equivalent to a logical notation where we can assert properties of strings in a systematic way that greatly simplifies the process of defining the types of regular languages and relations pertinent to natural language processing applications.

1. Notation

When speaking of regular languages, we denote alphabets with Δ , Γ , and Σ . We also make use of the standard extended regular expression operators: union ($\mathcal{X} \cup \mathcal{Y}$), concatenation ($\mathcal{X}\mathcal{Y}$), Kleene closure (\mathcal{X}^*), Kleene plus (\mathcal{X}^+), intersection ($\mathcal{X} \cap \mathcal{Y}$), complement ($\neg\mathcal{X}$), difference ($\mathcal{X} - \mathcal{Y}$), and asynchronous language product ($\mathcal{X} \parallel \mathcal{Y}$) (also called shuffle product). We follow the convention that individual symbols are represented in lower case, e.g. a , while arbitrary regular languages are calligraphic, e.g. \mathcal{A} , and reserve upper case letters to denote logical propositions, e.g. $S(\mathcal{X}, \mathcal{Y})$. We also use the standard logical quantifiers and connectives ($\exists x$), ($\forall x$), \neg , \vee , \wedge , \rightarrow , \leftrightarrow .

2. Method

We will first outline the reasoning informally and more concretely, and later introduce the notation more formally and generally.

Consider the effect of defining a language over an alphabet $\Delta = \Sigma \cup \Gamma$, where the alphabet is divided into two parts $\Sigma = \{a, b\}$ and $\Gamma = \{\bigcirc\}$ (our auxiliary alphabet), such that it contains exactly one instance of \bigcirc , i.e. $(\Sigma^* \bigcirc \Sigma^*)$, and then intersecting this language with a language that contains the \bigcirc -symbol, and finally deleting the \bigcirc

symbol from the language. Let us call this auxiliary symbol removal operation $\Pi(\mathcal{L})$. For example:

$$\Pi((\Sigma^* \bigcirc \Sigma^*) \cap (\Delta^* \bigcirc a \Delta^*)) \quad (2)$$

Clearly, the end result in this example is the language over Σ^* that contains at least one a , i.e. another way of saying $(\Sigma^* a \Sigma^*)$. However, laid out in this fashion, we can see that separating the regular expression into two parts has brought about two independent statements with different semantics: the first one, $(\Sigma^* \bigcirc \Sigma^*)$ simply asserts the existence of exactly one symbol \bigcirc , while the second, $(\Delta^* \bigcirc a \Delta^*)$ asserts that there is a \bigcirc -symbol which is followed by an a . Informally, the first part says “there exists exactly one position called \bigcirc ,” and the second part: “some position called \bigcirc is followed by an a .”

In effect what we have achieved in intersecting these two statements and deleting the \bigcirc -symbol is a form of variable binding—the first regular expression being equivalent to existential quantification of a position in a string, or the “existence of a substring,” and the latter a proposition bound by the variable \bigcirc .

We can now expand the same idea, and replace the first part of the regular expression with $(\Sigma^* \textcircled{x} \Sigma^* \textcircled{x} \Sigma^*)$ (for the sake of clarity in notation, let us replace the \bigcirc with \textcircled{x} in the auxiliary alphabet Δ , to make clear that our auxiliary symbol says something about a letter variable which we shall call x). We are now defining the language over Δ^* that contains exactly two symbols \textcircled{x} . The purpose of the two \textcircled{x} -symbols is to delineate two positions in a string x , the starting and the ending position (the usefulness of this will become clear later). Let us call the combined effect of this regular expression and of removing the auxiliary symbols the *regular expression equivalent* of $(\exists x)$, i.e. $\Pi(\Sigma^* \textcircled{x} \Sigma^* \textcircled{x} \Sigma^*)$. In intersecting this language before removal of the auxiliary symbols with any regular language φ (that may or may not contain \textcircled{x}) we can achieve a propositional sentence $(\exists x)(\varphi)$.

To continue with this idea: what about the possible propositions in φ ? In modelling of the open statements φ , the simplest desirable proposition would be one with the semantics that a substring is a member of some language \mathcal{L} , i.e. $x \in \mathcal{L}$. Over Δ^* the regular expression $(\Delta^* \textcircled{x} \mathcal{L} \textcircled{x} \Delta^*)$ describes precisely this circumstance: “there exists a substring $\textcircled{x} \mathcal{L} \textcircled{x}$.” Another useful statement to have would be a kind of successor-of-relationship $S(t_1, t_2) \text{---} t_1$ is immediately succeeded by t_2 —where the terms could either be arbitrary languages or variables. This translates naturally to $(\Delta^* t_1 t_2 \Delta^*)$, for example, $S(x, \mathcal{A})$ would be rendered as $(\Delta^* \textcircled{x} \Delta^* \textcircled{x} \mathcal{A} \Delta^*)$.

Since we have seen that $(\exists x)$ in our still tentative logic over strings can be modelled by $\Pi(\Sigma^* \textcircled{x} \Sigma^* \textcircled{x} \Sigma^*)$, and since a universally quantified proposition $(\forall x)(\varphi)$ is equivalent to $\neg(\exists x)\neg(\varphi)$, the regular expression equivalent of a universally quantified statement is:

$$(\forall x)(\varphi) \equiv \neg \Pi((\Sigma^* \textcircled{x} \Sigma^* \textcircled{x} \Sigma^*) \cap \neg(\varphi)) \quad (3)$$

We are now in a position to put together a complete logical sentence. For example, the sentence:

$$(\forall x)(x \in \mathcal{A} \rightarrow S(x, \mathcal{B})) \quad (4)$$

would describe the language where every instance of a member of language \mathcal{A} is immediately followed by a string from language \mathcal{B} . In translating the open statements to regular expressions, we make use of the conditional laws of statement logic, where $(P \rightarrow Q) \Leftrightarrow (\neg P \vee Q)$, and we find the equivalent regular expression following the above scheme:

$$\overbrace{\neg \Pi((\Sigma^* \textcircled{x} \Sigma^* \textcircled{x} \Sigma^*))}^{(\forall x)} \cap \neg(\neg(\overbrace{(\Delta^* \textcircled{x} \mathcal{A} \textcircled{x} \Delta^*)}^{x \in \mathcal{A}} \cup \overbrace{(\Delta^* \textcircled{x} \Delta^* \textcircled{x} \mathcal{B} \Delta^*)}^{S(x, \mathcal{B})}))$$

2.1. Variables

So far we have only assumed propositions quantified by a single variable x . Naturally, we will want to extend this to an arbitrary number of variables. This requires some book-keeping on the part of the alphabets. Suppose we have a sentence:

$$(\forall x)(\exists y)(\varphi) \quad (5)$$

Now, it will not do to define $(\exists y)$ as $(\Sigma^* \textcircled{y} \Sigma^* \textcircled{y} \Sigma^*)$ for the simple reason that this precludes the existence of \textcircled{x} symbols (as \textcircled{x} is not a symbol of Σ). So, with several symbols, we need the ability to describe “any symbol in Δ except \textcircled{y} ,” to ensure that we allow other auxiliary symbols in the regular expression equivalent of $(\exists y)$. This is of course easy to describe as a regular language $(\Delta - \textcircled{y})$, and as a shorthand and to keep the notation clean we shall say Δ_y signifies precisely this: any symbol in the alphabet Δ except \textcircled{y} . Hence, a construction such as

$$(\forall x)(\exists y)(\varphi) = \neg(\exists x)\neg((\exists y)(\varphi)) \quad (6)$$

becomes:

$$\neg \Pi((\Delta_x^* \textcircled{x} \Delta_x^* \textcircled{x} \Delta_x^*) \cap \neg \Pi((\Delta_y^* \textcircled{y} \Delta_y^* \textcircled{y} \Delta_y^*) \cap (\varphi))) \quad (7)$$

Until now, we have said little about the operation $\Pi(\mathcal{L})$, except that it deletes the symbols in our “variable alphabet” Γ from the language \mathcal{L} .¹ Again, in order to keep the notation uncluttered, we shall define $\Pi(\mathcal{L})$ as a *dynamic* operation, that also changes the alphabet Γ , shrinking it by one symbol, which is the symbol that is currently being removed. This operation is crucial for the possible language complements that need to be taken in the process of eliminating several quantifiers. In the above example (7), for instance, the innermost Π -operation deletes the symbol \textcircled{y} from the language and removes the symbol \textcircled{y} from Γ , leading to that the following complement is taken with respect to

¹From a formal language perspective, this is simply a substitution $f(\Gamma) = \epsilon$, or, from an automaton perspective, a replacement of transitions containing symbols from Γ with ϵ -transitions.

an alphabet Δ (recall that $\Delta = \Gamma \cup \Sigma$) that only contains one auxiliary $\{\textcircled{x}\}$. Likewise, after the outermost Π operation, $\Delta = \Sigma$, since all auxiliaries have now been purged from the auxiliary alphabet. This operation could be described non-dynamically, but at the cost of much lengthier expressions and without contributing to the clarity of the operation.

2.2. Propositions

We are naturally not restricted to the propositions developed so far—in fact any subset of the language Δ^* is a proposition.

Since a proposition, such as $x \in \mathcal{L}$, i.e. $(\Delta^* \textcircled{x} \mathcal{L} \textcircled{x} \Delta^*)$ may contain sublanguages where no variable symbols occur—in this example \mathcal{L} may be such a language—care must be taken to ensure that other variables can freely occur within the regular expression equivalent of the proposition. Hence, propositions should in general be augmented with freely interspersed symbols from Γ , our marker alphabet. The proposition $x \in \mathcal{L}$ then becomes $(\Delta^* \textcircled{x} \mathcal{L} \textcircled{x} \Delta^*) \parallel \Gamma^*$.²

For example, since we can now extend the notation with any proposition, we might want to define $S(t_1, t_2)$ as n -ary, instead of a two-place predicate (which will save much ink and instantiation of variables in some constructions). It is easy to see that $S(t_1, \dots, t_n) \equiv (\Delta^* t_1 \dots t_n \Delta^*) \parallel \Gamma^*$, where $t_i = \textcircled{x} \Delta^* \textcircled{x}$ if the term t_i is a variable, and simply \mathcal{L}_i if t_i is a language constant. For example, $S(\mathcal{L}, x, \mathcal{R})$, then becomes:

$$(\Delta^* \mathcal{L} \textcircled{x} \Delta^* \textcircled{x} \mathcal{R} \Delta^*) \parallel \Gamma^* \quad (8)$$

2.3. Interim summary

We now have a construction method by which our proposed logical notation can be systematically converted to regular expressions, and hence to finite-state automata.³ In particular, new propositions can be introduced in a fairly straightforward way, and we shall do so whenever needed in the upcoming examples. The basic construction together with basic propositions is summarized in Table 1.

We can now proceed to tackle a selection of difficult regular language problems and illustrate their solution through the notation developed here.

²This would of course be equivalent to $(\Delta^* \textcircled{x} (\mathcal{L} \parallel \Gamma^*) \textcircled{x} \Delta^*)$, which may be more efficient to compile because of less non-determinism in the intermediate results: if \mathcal{L} contains no symbols from Γ , which should be the case, then allowing symbols from Γ to freely occur within strings from \mathcal{L} will not introduce nondeterminism in the automaton construction. However, for the sake of generality, we will simply say that a proposition P shall be implemented as above, with symbols from Γ occurring anywhere, i.e. $P \parallel \Gamma^*$.

³The overall approach is somewhat similar to classical methods of converting sentences if first-order logic of one successor FO[<] and monadic second-order logic MSOL[S] to finite-state automata [9,10]. There are two crucial differences, however: a) classical methods employ a joint alphabet of symbols and variables represented as vectors, while we entirely divorce the variable alphabet from our symbol alphabet, and b) we treat variables semantically as denoting substrings with a beginning and an end, rather than integers representing positions in a string. This approach, we believe, confers much more conciseness in notation, and the advantage of a simple way of defining new predicates whenever necessary, as well as being compilable into automata using existing operations of finite-state toolkits. [11] also develops a logical formalism based on the classical methods mentioned above and applies it to language processing; however, this relies on a separate compiler from MSOL[S]-logic, and requires that extra predicates be defined in terms of primitive propositions, rather than allowing intermixing regular expressions and logical statements.

Logical notation	R.E. equivalent	Notes
$(\exists x)(\varphi)$	$\equiv \Pi((\Delta_x^* \textcircled{x} \Delta_x^* \textcircled{x} \Delta_x^*) \cap (\varphi))$	$\Delta_x = (\Delta - \textcircled{x})$
$(\forall x)(\varphi)$	$\equiv \neg \Pi((\Delta_x^* \textcircled{x} \Delta_x^* \textcircled{x} \Delta_x^*) \cap \neg(\varphi))$	
$x \in \mathcal{L}$	$\equiv (\Delta^* \textcircled{x} \mathcal{L} \textcircled{x} \Delta^*) \parallel \Gamma^*$	
$S(t_1, \dots, t_n)$	$\equiv (\Delta^* t_1 \dots t_n \Delta^*) \parallel \Gamma^*$	$t_i = \textcircled{v_i} \Delta^* \textcircled{v_i}$ if t_i is a variable x_i
$x = y$	$\equiv (\Delta^* (\textcircled{x} \parallel \textcircled{y}) \Delta^* (\textcircled{x} \parallel \textcircled{y}) \Delta^*) \parallel \Gamma^*$	

Table 1. Table summarizing the logical notation and their the regular expression equivalents. We assume the alphabets Γ and Σ , where Γ is the marker alphabet that contains the variable symbols under quantification, such as \textcircled{x} , \textcircled{y} , etc. Collectively, the two alphabets together are denoted Δ , i.e. $\Delta = \Gamma \cup \Sigma$. The operation $\Pi(\mathcal{L})$ deletes the currently quantified variable symbol from \mathcal{L} , and removes it from Γ .

2.4. An example construction

Returning to the the example construction of which a standard regular expression was given in Eq. (1), that of a language that contains only one factor from some arbitrary regular language \mathcal{L} ; in our logical notation, we could express this as:

$$(\exists x)(x \in \mathcal{L} \wedge (\forall y)(y \in \mathcal{L} \rightarrow x = y)) \quad (9)$$

Here we need a way to model the proposition $x = y$ for some variables x and y . This circumstance is captured by the language where both \textcircled{x} and \textcircled{y} markers share the same positions (see table 1).

Again, using the fact that $(P \rightarrow Q) \iff (\neg P \vee Q)$, and following the translation method given we get the following regular expression:

$$\Pi((\Delta_x^* \textcircled{x} \Delta_x^* \textcircled{x} \Delta_x^*) \cap (\alpha \cap \neg \Pi((\Delta_y^* \textcircled{y} \Delta_y^* \textcircled{y} \Delta_y^*) \cap \neg(\neg \beta \cup \gamma)))) \quad (10)$$

where:

$$\begin{aligned} \alpha &= (\Delta^* \textcircled{x} \mathcal{L} \textcircled{x} \Delta^*) \parallel \textcircled{x}^* \\ \beta &= (\Delta^* \textcircled{y} \mathcal{L} \textcircled{y} \Delta^*) \parallel (\textcircled{x} \cup \textcircled{y})^* \\ \gamma &= (\Delta^* (\textcircled{x} \parallel \textcircled{y}) \Delta^* (\textcircled{x} \parallel \textcircled{y}) \Delta^*) \parallel \Gamma^* \end{aligned}$$

It should be noted that there is much room for optimization in this particular construction: for instance, it is obvious that the shuffle product is unnecessary in α and partly so in γ , etc., however, we represent them explicitly here to follow the construction method mechanically. In general, depending on the nature of the propositions and the formula, some steps can be optimized or omitted to avoid unwanted nondeterminism in the intermediate stages of automaton construction.

3. Applications

3.1. Context restriction

Context restriction over arbitrary regular languages is an operation discussed in [12,8]. The notation is as follows:

$$\mathcal{A} \Rightarrow \mathcal{B}_1 _ \mathcal{C}_1, \dots, \mathcal{B}_n _ \mathcal{C}_n \quad (11)$$

The intended semantics is that a context restriction statement of the format above defines the language where every instance of a substring from the language \mathcal{A} is surrounded by some pair \mathcal{B}_i and \mathcal{C}_i . This language is quite challenging to capture through standard regular expression operators, as seen in the solution given in [12].

The language of context restriction translates very naturally into a logical notation: if x is a substring that is a member of language \mathcal{A} , then x is the successor of a string from \mathcal{B} and a string from \mathcal{C} is the successor of x . Employing the n -ary successor-of predicate introduced earlier, this becomes:

$$(\forall x) \left(x \in \mathcal{A} \rightarrow (S(\mathcal{B}_1, x, \mathcal{C}_1) \vee \dots \vee S(\mathcal{B}_n, x, \mathcal{C}_n)) \right) \quad (12)$$

and can be translated into a regular expression and a finite automaton exactly as described above.

3.2. Two-level rules

A two level grammar defines a subset of the language Σ_f^* , where Σ_f is the set of *feasible pairs*, defined in advance. The set Σ_f^* is constrained by the use of statements involving four operators: $a : b \Rightarrow l _ r$ (saying the symbol $a : b$ is only permitted between l and r), $a : b \Leftarrow l _ r$ (which says a symbol a occurring between the languages l and r must be realized as b), and $a : b / \Leftarrow l _ r$ (saying $a : b$ is never allowed between l and r) [13]. The notation $a : b \Leftarrow l _ r$ is a shorthand for the conjunction between the first two types of rules.

The feasible pairs Σ_f are also assumed to include every symbol pair occurring in some statement in the collection of grammar rules on the left-hand side.

Compiling a collection of such rules into a finite-state automaton (or transducer) is a non-trivial task, particularly for right-arrow rules with multiple contexts. However, each of the rule types are quite comfortably expressible in the logical notation proposed:

$$\begin{aligned} a : b \Rightarrow l _ r &\equiv (\forall x)(x \in a : b \rightarrow S(l, x, r)) \\ a : b \Leftarrow l _ r &\equiv \neg(\exists x)(x \in a : \bar{b} \wedge S(l, x, r)) \\ a : b / \Leftarrow l _ r &\equiv \neg(\exists x)(x \in a : b \wedge S(l, x, r)) \end{aligned}$$

There is the additional practice [3,2] that right-arrow rules with multiple contexts are allowed, are separated by commas, and are interpreted disjunctively: i.e. one of the contexts must hold for the symbol pair $a : b$ to be legal. For right-arrow rules, this prompts exactly the same solution as for context restriction above:

$$(\forall x)(x \in a : b \rightarrow S(l_1, x, r_1) \vee \dots \vee S(l_n, x, r_n)) \quad (13)$$

For left-arrow rules and disjunctive multiple contexts, the logical specification is:

$$\neg(\exists x)(x \in a : \bar{b} \wedge (S(l_1, x, r_1) \vee \dots \vee S(l_n, x, r_n))) \quad (14)$$

that is, in every context $l_i _ r_i$, a must be realized as b .

In essence, the above is a complete compilation algorithm and logical specification for two-level rules, if translated to regular expressions through the method presented above. The collection of individual rules are assumed to be intersected with each other and the set of feasible pairs. Hence, we get that a two-level grammar \mathcal{G} can be compiled as:

$$\mathcal{G} = \Sigma_f^* \cap \mathfrak{R}$$

where \mathfrak{R} is the intersection of the individual rules compiled through the notation presented here.

3.3. String-to-string two-level rules vs. replacement rules

Many proposals have been put forth regarding the conversion of so-called phonological rewrite rules into finite-state transducers. This includes [4], [7], [14], [6], inter alia. We shall here consider the construction of transducers that implement “replacement rules,” as defined in [2]. However, in order to simultaneously construct transducers and clearly define the semantics of these replacement rules, we shall build them as an extension to two-level rules and interpret them as two-level correspondence rules augmented with specific *conflict resolution strategies*.

3.3.1. Replacement rules

[2] define a set of replacement rules of the format:

$$A \text{ op } B \text{ dir } L _ R$$

where op is one of \rightarrow (replace), $@\rightarrow$ (replace leftmost, longest-match), $@>$ (leftmost shortest-match), and dir one of $|$ (upper-side context), $//$ (left context holds on lower side, right on upper), $\backslash\backslash$ (left context holds on upper side, right on lower), $\backslash/$ (both contexts hold on lower side). Replacement op may also be optional by surrounding it with parentheses, e.g. (\rightarrow) . Multiple replacement rules may also apply in parallel, which is indicated by separating the different rules with „,“, although in this case op and dir must be identical for all parallel rules.⁴

⁴This is based on an observation from the reference implementation of these operators, xfst 2.10.9.

The proximity between a replacement rule $a \rightarrow b \mid l _ r$ and a two-level rule $a : b \Leftrightarrow l : _ r :$ has been noted by [5]. Also, an “optional” replacement rule (\rightarrow) seems to correspond somewhat to two-level right-arrow (\Rightarrow) rules. What stands in the way of this analogy is that replacement rules are assumed to describe a relation where the arguments are *any* regular language, while the two-level paradigm is restricting symbol-to-symbol correspondences. To bring these two formalisms closer together, we shall therefore as a first step extend the two-level formalism to allow any regular languages as arguments, and see that, under a certain interpretation, the formalisms of parallel replacement rules and string-to-string two-level rules with a conflict resolution strategy describe exactly the same relations. Through this approach we also give a clear logical definition of the two using the formalism presented in this paper.

3.3.2. String-to-string two-level rules

In what follows, we shall assume the regular semantics of two-level rules, with the addition that in statements such as:

$$A : B \text{ op } L _ R$$

where op is one of $\Leftrightarrow, \Leftarrow, \Rightarrow$, A and B may be *any* regular language, while L and R may be a correspondence between any two regular languages, i.e. $L_1 : L_2$. For example:

$$a^+ : x \Leftrightarrow a : \Sigma^* _ b : \Sigma^*$$

Also, we abandon the idea of constraining a set of feasible pairs, and replace this with the notion of *feasible string pairs*, which is a subset of $(\Sigma^* \times \Sigma^*)$, which includes all single-symbol identity pairs, as well as every language pair $A : B$ used in a two-level rule, i.e. a grammar with a single rule like $a : b^+ \Leftrightarrow x : \Sigma _ x : \Sigma$ is a constraint over the feasible string pairs $(Id(\Sigma) \cup a \times b^+)^*$.⁵

3.3.3. Rule conflicts

As is well known for writers of two-level grammars, rules may conflict with each other in the sense that one rule blocks the application of another rule, or two rules are mutually contradictory. These are classifiable as both left-arrow conflicts and right-arrow conflicts. An example of a left-arrow conflict, is given by the pair of rules:

$$a : b \Leftarrow l _ r \qquad a : c \Leftarrow l _ r$$

Obviously, there cannot exist a string that contains l followed by an a on the lexical side, and r , where the a is realized as both b and c on the surface side. A right-arrow conflict can be illustrated by the pair of rules:

$$a : b \Rightarrow l _ r \qquad a : b \Rightarrow c _ d$$

⁵See the appendix for a suggested encoding of regular relations as a regular language at the automaton level.

Here, the two rules state that the pair $a : b$ is only allowed in two different contexts: l_r and c_d , in effect disallowing $a : b$ everywhere.

For practical purposes, automatic conflict resolution has usually focused on two strategies: either giving automatic precedence to one of two rules, or giving precedence to the rule which is more ‘specific’ in its context specification (related to the notion called ‘disjunctive ordering’ in serial rewrite-rule phonology).

However, in generalizing two-level rules to arbitrary strings, there is the additional possibility that a rule may be in conflict with itself.⁶ Consider the rule:

$$a^+ : b \Leftrightarrow _ \quad (15)$$

i.e., a^+ may and must be realized as b everywhere. Two hypothetical string pairs that at first glance may seem feasible are:

$$\begin{array}{cc} a & a \\ b & b \end{array} \qquad \begin{array}{cc} a & a \\ b & 0 \end{array}$$

However, both possibilities are ruled out: in the first case aa (a member of a^+ is corresponding to bb (not a member of the language b), and in the second case the latter a (a member of a^+) is realized as 0 (not a member of b). Hence, any input involving more than one consecutive a has no feasible realization: the rule is in left-arrow conflict with itself.

Also, consider the rule:

$$a^+ : b \Leftrightarrow l : \Sigma^* _ \quad (16)$$

and the two pairs:

$$\begin{array}{cc} l & a & a \\ l & b & a \end{array} \qquad \begin{array}{cc} l & a & a \\ l & b & 0 \end{array}$$

Here, the leftmost correspondence is ruled out because a substring aa corresponds to ba , while the rule says a^+ must be realized as b everywhere. Similarly for the rightmost case: the last a is a member of the language a^+ , but is realized as zero, not b .

Herein lies a crucial difference between the expected behavior of replacement rules: a replacement rule

$$a^+ \rightarrow b \mid l _ \quad (17)$$

will accept both correspondences described. It seems a two-level formalism based on string-to-string constraints becomes less useful than symbol-to-symbol constraints because of this stringency.

⁶Which is indeed also possible in symbol-to-symbol two-level rules, but only with epenthesis rules, i.e. rules of the type $0 : a \Leftarrow l_r$. This and other epenthesis-related cases will be discussed in the appendix.

If we wanted to pursue the analogy between replacement rules and string-to-string two-level rules, there is also the possibility of writing replacement rules that apply in “parallel,” e.g.

$$l \ a \ r \rightarrow l \ a \ l \quad | \quad _ \ _ \ a \rightarrow b \quad | \quad l \ _ \ r \quad (18)$$

The way the parallel rules are defined, the two correspondences:

$$\begin{array}{ll} l \ a \ r & l \ a \ r \\ l \ a \ l & l \ b \ r \end{array}$$

are allowed. However, for a similar set of two-level constraints

$$lar : lal \Leftrightarrow _ \qquad a : b \Leftrightarrow l : _ r :$$

neither one is allowed, since they are in mutual conflict.

3.3.4. Conflict resolution

The above data suggest a strategy for automatic conflict resolution to make string-based two-level rules useable: if we would like to define transducers that *always* give some output, regardless of the input, and, at the same time, not give certain rules arbitrary precedence over others, the only feasible strategy is to have rules always yield to other rules if they are in conflict. In the previous example, we will accept both *lal* and *lbr* as outputs for the input *lar*, because in the two cases, rule 1 should yield to rule 2, and vice versa, mutually.

In formalizing this using our logic notation, we want to say that a rule (left-arrow or right-arrow) must hold, except if another rule permits a different correspondence in parts of the same center. To this end, we shall need a new predicate in addition to the ones already established: call this two-argument predicate OL (for ‘overlaps’), with the semantics that a proposition $OL(t_1, t_2)$ is true *iff* the positions of strings t_1 and t_2 share at least one symbol:

$$\begin{array}{c} x \\ \underbrace{\hspace{1cm}} \\ \dots\dots\dots \\ \underbrace{\hspace{1cm}} \\ y \end{array}$$

We can now add a statement to the previous definition of (\Rightarrow) to illustrate this method of conflict resolution:

$$\begin{array}{l} A : B \Rightarrow L _ R \equiv \\ (\forall x)(x \in A : B \rightarrow S(L, x, R) \vee (\exists y)(OL(x, y) \wedge y \in A : B \wedge S(L, y, R))) \end{array} \quad (19)$$

that is, every $A : B$ must be surrounded by L and R , except in the case that an $A : B$ is a substring of another $A : B$ which in turn is surrounded by L and R .

However, this only solves the case where a single rule may be in conflict with itself, such as:

$$a^+ : b \Leftrightarrow _$$

In general, we would like to enforce a right-arrow constraint only in the case that there is *no other* rule applicable for a given substring of correspondences. Assume we have a set of right-arrow rules $\{\mathfrak{R}_1, \dots, \mathfrak{R}_n\}$ that consist of components $A_i : B_i, L_i, R_i$, and we want to express the idea that all right-arrow correspondence requirements $A_i : B_i$ must hold, except if some other correspondence (including one permitted by the rule at hand itself) is legal within the same region $A_i : B_i$.

$$\bigwedge_{i=0}^n (\forall x_i)(x_i \in A_i : B_i \rightarrow S(L_i, x_i, R_i)) \quad (20)$$

$$\bigvee_{j=0}^n (\exists y_j)(OL(x_i, y_j) \wedge y_j \in A_j : B_j \wedge S(L_j, y_j, R_j))$$

There is an equivalence between replacement rules of the directional type and two-level string-to-string rules with the conflict resolution method, according to the following pattern:

$A \rightarrow B \mid \mid L _ R$	$A : B \Leftrightarrow L : \Sigma^* _ R : \Sigma^*$
$A \rightarrow B \ / \ / L _ R$	$A : B \Leftrightarrow \Sigma^* : L _ R : \Sigma^*$
$A \rightarrow B \ \backslash \backslash L _ R$	$A : B \Leftrightarrow L : \Sigma^* _ \Sigma^* : R$
$A \rightarrow B \ \backslash / L _ R$	$A : B \Leftrightarrow \Sigma^* : L _ \Sigma^* : R$

However, replacement rules are somewhat more restricted since contexts are only specified on one side of a relation.

If we generalize and apply the same conflict resolution method to \Leftarrow , the two formalisms become equivalent. That is, the formula (20) together with an identical conflict resolution-enhanced formula for \Leftarrow will produce transducers that function exactly as parallel replacement rules. Right-arrow rules by themselves produce so-called optional replacement rules (\rightarrow) .

3.4. Modes of application

There is additionally the type of replacement rule that is constrained by length-of-match and direction (either left-to-right or right-to-left). So for instance, a leftmost longest-match rule like:

$$a^+ @ \rightarrow x \quad (21)$$

will map a to x , aa to x , etc [7].

Visualizing the “directionality” as either left-to-right or right-to-left conjures up a procedural image of the string correspondence, which in a logical formalism is difficult to approach. But there is a static way of looking at the formalism only in terms of string correspondences. “Longest match” in a rule that constrains a pairing of A and B (perhaps between L and R) clearly implies that a correspondence $A : B$ may not occur if that same string $A : B$ starts at the position a longer string $A : \neg B$ starts. That is, the configuration:

$$\begin{array}{ccc} L & A : \neg B & R \\ \underbrace{\quad \quad \quad} & \underbrace{\quad \quad \quad} & \underbrace{\quad \quad \quad} \\ \dots & \dots & \dots \\ \underbrace{\quad \quad \quad} & \underbrace{\quad \quad \quad} & \underbrace{\quad \quad \quad} \\ L & A : B & R \end{array}$$

should be disallowed. Returning to our original definition of a right-arrow rule (disregarding for the moment what was said previously about conflict resolution), we need to restrict the right-arrow rule as follows:⁷

$$(\forall x)(x \in A : B \rightarrow S(L, x, R) \wedge \neg(\exists y)(y \in A : \neg B \wedge (x_b = y_b) \wedge (y_e > x_e) \wedge S(L, y, R))) \quad (22)$$

This defines the language where $A : B$ is only allowed between L and R , but that $A : B$ is additionally disallowed in the circumstance described above: if there also exists a substring y which starts where x starts, ends later than x and y could potentially be a legal $A : B$ correspondence, but is not.

The “leftmost” requirement is an additional constraint symmetrical to the longest-match requirement: we also want to disallow an $A : B$ whenever we can find an instance of $L A : \neg B R$ where the $A : \neg B$ portion overlaps with the position at hand and that begins earlier.

We therefore want to add the statement

$$\neg(\exists y)(y \in A : \neg B \wedge S(L, y, R) \wedge OLL(y, x)) \quad (23)$$

where $OLL(y, x)$ is a proposition “overlaps-to-the-left,” describing situations such as:⁸

$$\dots \textcircled{y} \dots \textcircled{x} \dots \textcircled{y} \dots \textcircled{x} \dots$$

The cases of shortest-match and right-to-left directionality are entirely symmetrical: in shortest-match we disallow an $A : B$ that contains $A : \neg B$ starting at the same position, while right-to-left requires the proposition overlaps-to-the-right.

⁷We temporarily introduce two new propositions here; it is easy to see that $(x_s = y_s)$, the substring x begins where the substring y does, and $(x_e > y_e)$, the substring x ends later than y can be constructed as a regular expression over Δ —the former is the language where the first \textcircled{x} is adjacent to the first \textcircled{y} -symbol, while the latter is the language where the second \textcircled{x} occurs later than the second \textcircled{y} -symbol (with at least one symbol from Σ intervening).

⁸i.e. $(\Delta^* \textcircled{y} \Delta^* \textcircled{x} \Delta^* \textcircled{y} \Delta^* \textcircled{x} \Delta^*)$ —this also permits the situation where the two variables denote exactly the same substring. This is a regular expression over simple strings, not correspondences. See the appendix on how to modify this to handle correspondences in our encoding.

Again, if we add the conflict resolution method outlined in the previous section to these modifications of the right-arrow rule, the semantics of the collection of replacement rules, parallel replacement rules as well as directed replacement are captured through the notation here, and are directly compilable into finite-state automata/transducers.

4. Conclusion

We have presented an extension to the formalism of regular expressions, which we call regular predicate logic. It systematizes the prevalent use of auxiliary symbols in defining complicated languages in a way that is notationally clear and can be intermixed with standard regular expressions. In particular, the propositions of our regular predicate logic are freely extendable and it is assumed that one can take advantage of other finite-state calculus operators in defining new predicates.

We have also demonstrated how the notation can be used to systematically define other formalisms used in natural language processing applications; two-level rules and replacement rules. We believe the new notation brings a level of transparency to the definition of other complex regular expression operations.

It is interesting to note that [4], in defining what they call “if-P-then-S($\mathcal{L}_1, \mathcal{L}_2$)” (the language where every string from \mathcal{L}_1 is followed by some string from \mathcal{L}_2) as $\neg(\Sigma^* \mathcal{L}_1 \neg(\mathcal{L}_2 \Sigma^*))$, point out an intuition that the “double complementation in the definitions ... and also in several other expressions ... constitutes an idiom for expressing universal quantification.” However, in our formalism it is the *combination* of a specific type of use of auxiliary symbols together with a double negation that constitutes an idiom for universal quantification. The double negation (taken with respect to different alphabets) then becomes an artifact of the definition of universal quantification in terms of existential quantification and the construct where variable binding is achieved through intersection:

$$(\forall x)(\varphi) \equiv \neg \Pi((\Delta_x^* \otimes \Delta_x^* \Delta_x^*) \cap \neg(\varphi))$$

4.1. Further work

The examples given in this paper are in no way meant to be exhaustive of the potential applications of the formalism. Examples of possible further work include:

- Investigation other formalisms in terms of the logical notation developed here. For instance, [15] and [1] treat Optimality Theory fundamentally through inserting violation markers, filtering them, and removing them. This can probably be interpreted within the notation at hand, where different violations are treated as different variables, which would yield a more static, logical description of OT. This would probably require an introduction of limited second-order predication together with a “cardinality” predicate (which would go beyond finite-state means in the general case), in order to keep track of the number of violations of constraints.

- Multi-tiered constraint systems could probably be described through the logical notation developed here. In essence, the string generalization of two-level grammars could probably be augmented from two to any number of tiers, where logical statements could be made within, between, and across tiers. This bears many similarities to autosegmental theories of phonology.
- The formalism itself could possibly be used as a sole basis for constructing natural language morphological analyzers, either through a 2-level or n-level representation.

5. Appendix

5.1. Multi-level encoding

In implementing string-to-string two-level rules and replacement rules, we have decided to not directly encode the relations as transducers, but rather as an even-length regular language $(\Sigma\Sigma)^*$ such that the odd numbered symbols represent the input and the even numbered symbols the output. Additionally the alphabet contains a special symbol 0 which represents ϵ (very similar to the “hard zero” in classical two-level implementations). A language over $(\Sigma\Sigma)^*$ can trivially be converted into a finite-state transducer by removing odd states in a path and creating symbol pairs from sequences, i.e. a sequence ab would become a single transition $a : b$. Also, any string pair $A : B$ mentioned in a rule is arranged in such a way that the symbol 0 is padded to the end of whichever the shorter string is to make an equal-length representation: $(abc) : d$ becomes $adb0c0$, etc. The initial “feasible language pairs” are then all doubled symbol sequences, in the example above: $(aa \cup bb \cup cc \cup dd \cup @@ \cup adb0c0)^*$, etc. The additional special symbol sequence @@ signifies an identity pair for any symbol not explicitly included in the alphabet through a left-hand side of a two-level rule.

Naturally, the logical compilation must be modified to accommodate the fact that we are dealing with symbol sequences where the symbols come in pairs: i.e. we need to also make sure the variable symbols occur at positions before even-numbered symbols from Σ so that propositions that need to refer either to the input side or the output side are consistent. Hence, $(\exists x)$ in this two-level encoding becomes:

$$((\Delta_x \Delta_x)^* \textcircled{x} \textcircled{x} (\Delta_x \Delta_x)^* \textcircled{x} \textcircled{x} (\Delta_x \Delta_x)^*)$$

and similarly for encoding the propositions.

5.2. Epenthesis rules

Epenthesis rules, exemplified in two-level grammars through rules where the left-hand side is $0 : a$, or in replacement rules such as $0 \rightarrow a$ have a special status owing to the fact that the empty string ϵ , from a formal point of view, occurs an unbounded number of times within any string in a language. In fact, all approaches to treating such statements need to make explicit decisions on the semantics of epenthesis rules. Replace rules (as in [2]) come in two varieties: $0 \rightarrow L$, and $[. . .] \rightarrow L$ with different semantics. The former is interpreted as an optional rule (inserting optionally the language L in the speci-

fied context), while the latter is an obligatory rule with the restriction that ϵ is interpreted as occurring only and exactly once between each symbol in Σ^* . These are arbitrary decisions motivated by the prevalence of epenthesis rules in phonology and the need to model such patterns through finite-state means. Their behavior can be modelled readily through the logic notation presented above.

References

- [1] Dale Gerdemann and Gertjan van Noord. Approximation and exactness in finite state optimality theory. In Alain Thériault, Jason Eisner, and Lauri Karttunen, editors, *Proceedings of the Fifth Workshop of the ACL Special Interest Group in Computational Phonology*, 2000.
- [2] Kenneth Beesley and Lauri Karttunen. *Finite-State Morphology*. CSLI, Stanford, 2003.
- [3] Lauri Karttunen, Kimmo Koskenniemi, and Ronald M. Kaplan. A compiler for two-level phonological rules. In M. Dalrymple, R. Kaplan, L. Karttunen, K. Koskenniemi, S. Shao, and M. Wescoat, editors, *Tools for Morphological Analysis*. CSLI, Palo Alto, CA, 1987.
- [4] Ronald M. Kaplan and Martin Kay. Regular models of phonological rule systems. *Computational Linguistics*, 20(3):331–378, 1994.
- [5] Lauri Karttunen. The replace operator. In Emmanuel Roche and Yves Schabes, editors, *Finite-State Language Processing*. MIT Press, 1997.
- [6] Andre Kempe and Lauri Karttunen. Parallel replacement in finite state calculus. *Proceedings of the 16th conference on Computational linguistics*, 2:622–627, 1996.
- [7] Lauri Karttunen. Directed replacement. In *Proceedings of the 34th conference on Association for Computational Linguistics*, pages 108–115, 1996.
- [8] Anssi Yli-Jyrä and Kimmo Koskenniemi. Compiling contextual restrictions on strings into finite-state automata. In *The Eindhoven FASTAR Days Proceedings*, 2004.
- [9] J. Richard Büchi. Weak second-order arithmetic and finite automata. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, 6:66–92, 1960.
- [10] C.C. Elgot. Decision problems of finite automata and related arithmetics. *Trans. Amer. Math. Soc.*, 98:21–51, 1961.
- [11] Nathan Vailllette. Logical specification of regular relations for nlp. *Natural Language Engineering*, 9(1):65–85, 2003.
- [12] Anssi Yli-Jyrä. Describing syntax with star-free regular expressions. In *11th EACL 2003, Proceedings of the Conference*, pages 379–386, 2003.
- [13] Kimmo Koskenniemi. Two-level morphology: A general computational model for word-form recognition and production. Publication 11, University of Helsinki, Department of General Linguistics, Helsinki, 1983.
- [14] Mehryar Mohri and Richard Sproat. An efficient compiler for weighted rewrite rules. In *Proceedings of the 34th conference on Association for Computational Linguistics*, pages 231–238, 1996.
- [15] Lauri Karttunen. The proper treatment of optimality theory in computational phonology. In *Finite-state Methods in Natural Language Processing*, pages 1–12, Ankara, 1998.

Making Finite-State Methods Applicable to Languages Beyond Context-Freeness via Multi-dimensional Trees

Anna KASPRZIK
University of Trier

Abstract. We provide a new term-like representation for multi-dimensional trees as defined by Rogers [1,2] which establishes them as a direct generalization of classical trees. As a consequence these structures can be used as input for finite-state applications based on classical term-based tree language theory. Via the correspondence between string and tree languages these applications can then be conceived to be able to process even some language classes beyond context-freeness.

Keywords. Finite-state methods, Multi-dimensional Trees, Regularization, Tree Adjoining Grammar

Introduction

It is well known that string languages that are recognizable by finite-state automata, the so-called regular languages, have a whole range of advantageous mathematical properties. However, due to the relatively restricted character of the latter, classes that lie beyond regularity are often more interesting for applications based on formal language theory, even if the devices processing these languages (e.g., grammars or automata) are significantly more complex. Obviously, it would be of considerable use if one could reunite the advantages of regular and less restricted language classes by finding a way to handle these processes via regular mechanisms without giving up any of the expressive power.

Several such regularization methods have indeed been formulated, and at least two of them have been shown to be of use in the field of linguistics. A very prominent linguistic application of formal language theory is the area of natural language processing, i.e., conceiving the strings formed by a natural language as a formal language in order to treat them automatically. Unfortunately, the study of certain phenomena (e.g., cross-serial dependencies in Dutch or Swiss German) showed that some of these string sets are not context-free. Joshi [3] claimed the least class of formal languages containing all natural language string sets to be situated between the context-free and the context-sensitive languages, and named it the class of *mildly context-sensitive languages*. The string sets generated by the grammar formalism defined by Joshi [3] himself, Tree Adjoining Grammar, prototypically fulfil all the necessary conditions for this class. TAG is considered the standard model for mild context-sensitivity and is the foundation of a considerable amount of current work in applied computational linguistics.

There are two methods of regularization for TAG (see [4]). Both methods are two-step approaches, i.e., they transform a TAG into a regular device (grammar or automaton) of some sort by representing its components in another shape and then reconstruct the intended objects from the objects generated or licensed by these devices via a simple process that can be carried out with regular means as well. One is based on an algebraic operation called Lifting (see [5]) which could be described as a way to write terms in a form that makes their internal structure more explicit, which, if the term is noted as a tree, has the side effect that all inner nodes are turned into leaves and thus become rewritable by substitution, which is a regular mechanism, and the other method (described by Rogers [1,2]) makes use of an additional dimension in space by representing the components of a TAG as three-dimensional trees which likewise has the consequence that all inner nodes are turned into leaves and can be expanded by substitution.

The theoretical foundation of the second method are *multi-dimensional trees*, which are structures built over tree domains of arbitrarily many dimensions. Just like ordinary two-dimensional trees, every multi-dimensional tree has a string associated with it, which is obtained by reducing the dimensions of the tree step by step to its leaves. The classes of string languages associated with the recognizable multi-dimensional tree languages ordered by number of dimensions form a (proper) infinite hierarchy properly contained in the context-sensitive class, with the classes of finite languages (associated with zero-dimensional point sets), regular languages (one-dimensional string sets), context-free languages (two-dimensional tree sets, as for the classical definition) and the mildly context-sensitive string languages generated by (non-strict) TAGs (three-dimensional tree sets, see [1,2]) as the first four steps. According to Rogers [1], this hierarchy coincides with Weir's Control Language Hierarchy [6].

It follows from these correspondences that by processing recognizable higher-dimensional descriptions of non-regular string languages instead of the string sets themselves, finite-state methods become applicable again, and with them all the advantages and results pertaining to regularity. This can be a valuable insight in the area of natural language processing, but also in other areas based on formal language theory, e.g., grammatical inference: For instance, just as Angluin's learning algorithm for regular string languages [7] has been adapted to regular tree languages [8,9], thereby making context-free string languages learnable (wrt the underlying learning model), this algorithm can be generalized to recognizable tree languages of arbitrarily many dimensions, making even string languages beyond context-freeness learnable in polynomial time (see [10]).

However, before such applications founded on formal tree languages can be generalized to arbitrarily many dimensions, there is a missing link to be provided: Most of them are not based on tree domains, as is Rogers' definition of multi-dimensional trees, but on the concept of trees as terms over a partitioned alphabet (the partitioning being induced by rank in the traditional case). In this paper we will give a new term-like representation for multi-dimensional trees, along with an adapted definition of finite-state automata for these structures, and prove the equivalence of the two notations. As a consequence multi-dimensional trees can be seen and used as a direct generalization of classical trees, and the full range of beneficial results for regular (tree) languages as known from formal language theory in the spirit of the Chomsky Hierarchy can be exploited.

1. Preliminaries

We presuppose familiarity with classical formal language theory (see for example [11]). We will give some basic notions regarding trees (see for example [12,13]).

A *ranked alphabet* is a finite set of symbols, each associated with a rank $n \in \mathbb{N}$. By Σ_n we denote the set of all symbols in Σ with rank n . Traditionally, every symbol has a single rank, but it is just as possible to admit several ranks for one symbol, as long as there is a maximal rank and the alphabet stays finite.

The set T_Σ of all trees over Σ is defined inductively as the smallest set of expressions such that $f[t_1, \dots, t_n] \in T_\Sigma$ for every $f \in \Sigma_n$ and all $t_1, \dots, t_n \in T_\Sigma$. t_1, \dots, t_n are the *direct subtrees* of the tree. The set $\text{subtrees}(t)$ consists of t itself and all subtrees of its direct subtrees. A subset of T_Σ is called a tree language.

Let \square be a special symbol of rank 0 (leaf label). A tree $c \in T_{\Sigma \cup \{\square\}}$ in which \square occurs exactly once is called *context*, the set of all contexts over Σ is denoted by C_Σ . For $c \in C_\Sigma$ and $s \in T_\Sigma$, $c[[s]]$ denotes the tree obtained by substituting s for \square in c . The depth of c is the length of the path from the root to \square .

A (*total, deterministic*) *bottom-up finite-state tree automaton (fta)* is a tuple $\mathcal{A} = (\Sigma, Q, \delta, F)$ where Σ is the ranked input alphabet, Q is the finite set of states, δ is the transition function assigning to every $f \in \Sigma_n$ and all $q_1, \dots, q_n \in Q$ a state $\delta(q_1 \dots q_n, f) \in Q$, and $F \subseteq Q$ is the set of accepting states. The transition function extends to trees: $\delta : T_\Sigma \rightarrow Q$ is defined such that if $t = f[t_1, \dots, t_n] \in T_\Sigma$ then $\delta(t) = \delta(\delta(t_1) \dots \delta(t_n), f)$. The language accepted by \mathcal{A} is $L(\mathcal{A}) = \{t \in T_\Sigma \mid \delta(t) \in F\}$. Such a tree language is called *regular*.

It is well known that the Myhill-Nerode theorem carries over to regular tree languages: Let $L \subseteq T_\Sigma$. Given two trees $s, s' \in T_\Sigma$, let $s \sim_L s'$ iff for every $c \in C_\Sigma$, either both of $c[[s]]$ and $c[[s']]$ are in L or none of them is. Obviously, \sim_L is an equivalence relation on T_Σ . The equivalence class containing $s \in T_\Sigma$ is denoted by $[s]_L$. The *index* of L is the cardinality of $\{[s]_L \mid s \in T_\Sigma\}$. The Myhill-Nerode theorem states that L is regular iff L is of finite index. It follows that for every fta \mathcal{A} , $L(\mathcal{A})$ is of finite index. Conversely, if a tree language is of finite index, we can easily build an fta \mathcal{A}_L recognizing L , with the states being the equivalence classes of L , $F = \{[s]_L \mid s \in L\}$, and, given some $f \in \Sigma_k$ and states $[s_1]_L, \dots, [s_k]_L$, $\delta_L([s_1]_L, \dots, [s_k]_L, f) = [f[s_1, \dots, s_k]]_L$. Moreover, this fta is the unique minimal fta recognizing L , up to a bijective renaming of states.

The Pumping lemma for regular tree languages (see [13] for a proof):

Lemma 1 *For any regular tree language $T \subseteq T_\Sigma$ there is a number $n \geq 1$ such that, if $t \in T_\Sigma$ has height $k \geq n$, then, for some $s \in T_\Sigma$ and $p, q \in C_\Sigma$, $t = q[[p[[s]]]]$ where p has depth ≥ 1 and $q[[\underbrace{p[[\dots p[[s]]] \dots}_{k \text{ times}}]] \in T_\Sigma$ for all $k \geq 0$.*

To conclude this section we give a definition for the grammar formalism Tree Adjoining Grammar, which was designed under linguistic considerations by Joshi ([3]) and was a main motivation for the study of multi-dimensional trees for Rogers. Rogers [1,2] defines *non-strict* TAGs as follows:

Definition 1 *A non-strict TAG is a pair $\langle E, I \rangle$ where E is a finite set of elementary trees in which each node is associated with a label from some alphabet, an SA constraint (a subset of E), and an OA constraint (Boolean valued). $I \subseteq E$ is a distinguished non-empty subset. Every elementary tree has a foot node.*

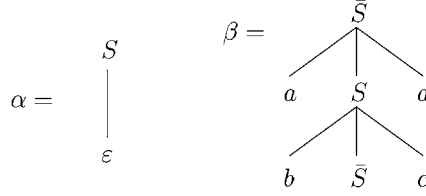


Figure 1. A TAG generating the (non-context-free) string language $a^n b^n c^n d^n$.

In a TAG, new trees can be built by *adjunction*: A node in a tree is replaced by another tree and the subtree formerly rooted at that node is attached to the foot node of the inserted tree. OA constraints state if adjunction is required or not, and SA constraints state which trees may be adjoined at that node.

Example 1 Let $G = \langle \{\alpha, \beta\}, \{\alpha\} \rangle$ be a TAG (over the alphabet $\{a, b, c, d, S\}$). α and β are given in Figure 1. Constraints at all inner nodes and the foot node of β (the leaf labeled with \bar{S}) are: $OA = 0$ and $SA = \{\beta\}$ for the ones without a bar, $OA = 0$ and $SA = \emptyset$ for the ones labeled with ' \bar{S} '. The bar stands for null adjunction, no adjunction is allowed at these nodes. G generates the (non-context-free) string language $a^n b^n c^n d^n$ for $n \geq 0$.

2. Multi-dimensional Trees and Automata

In this section we will introduce multi-dimensional trees and some related concepts as presented by Rogers [1,2].

Starting from a definition of ordinary trees based on two-dimensional tree domains, Rogers [1,2] generalizes the concept both downwards (to strings and points) and upwards and defines *labeled multi-dimensional trees* based on a hierarchy of multi-dimensional tree domains:

Definition 2 Let ${}^d 1$ be the class of all d th-order sequences of 1s: ${}^0 1 := \{1\}$, and ${}^{d+1} 1$ is the smallest set satisfying (i) $\langle \rangle \in {}^{d+1} 1$, and (ii) if $\langle x_1, \dots, x_l \rangle \in {}^{d+1} 1$ and $y \in {}^d 1$, then $\langle x_1, \dots, x_l, y \rangle \in {}^{d+1} 1$. Let $\mathbb{T}^0 := \{\emptyset, \{1\}\}$ (point domains). A $(d+1)$ -dimensional tree domain is a (finite) set of hereditarily prefix closed $(d+1)$ st-order sequences of 1s, i.e., $T \in \mathbb{T}^{d+1}$ iff ('.' representing concatenation)

- $T \subseteq {}^{d+1} 1$,
- $\forall s, t \in {}^{d+1} 1 : s \cdot t \in T \Rightarrow s \in T$,
- $\forall s \in {}^{d+1} 1 : \{w \in {}^d 1 \mid s \cdot \langle w \rangle \in T\} \in \mathbb{T}^d$.

A Σ -labeled \mathbb{T}^d (d -dimensional tree) is a pair $\langle T, \tau \rangle$ where T is a d -dimensional tree domain and $\tau : T \rightarrow \Sigma$ is an assignment of labels in the (non-partitioned) alphabet Σ to nodes in T . We will denote the class of all Σ -labeled \mathbb{T}^d as \mathbb{T}_Σ^d .

Every d -dimensional tree can be conceived to be built up from one or more d -dimensional *local trees*, that is, trees of depth at most one in their major dimension. Each of these smaller trees consists of a root and an arbitrarily large $(d-1)$ -dimensional “child tree” consisting of the root’s children (a formal definition of the

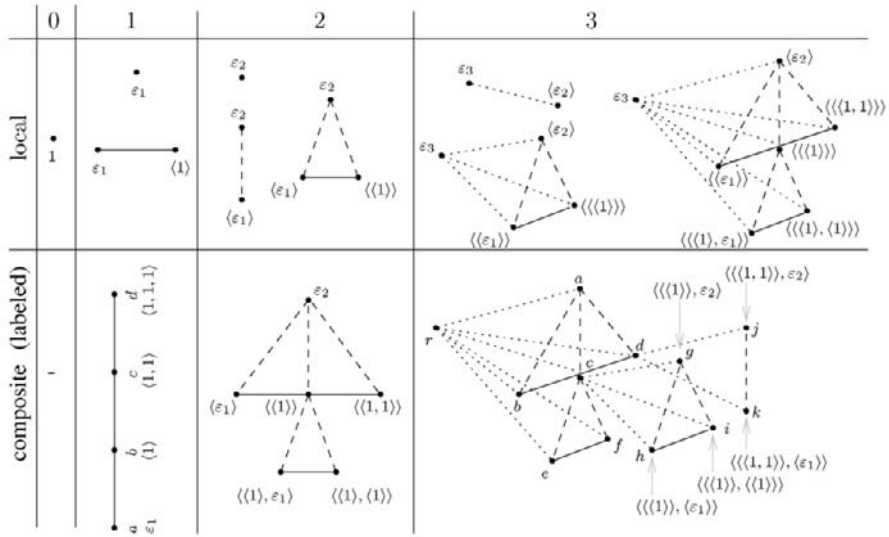


Figure 2. Local and composite trees for $d = 0, 1, 2, 3$

set $\mathbb{T}_{\Sigma}^{d,loc}$ of all local trees over some alphabet Σ would be for example $\mathbb{T}_{\Sigma}^{d,loc} = \{\langle T, \tau \rangle \mid \langle T, \tau \rangle \text{ is a } \Sigma\text{-labeled } \mathbb{T}d, \text{ and } \forall s \in T : |s| \leq 1\}$. Local strings (i.e., one-dimensional trees), for example, consist of a root and a point as its child tree. Local two-dimensional trees consist of a root and a string. Local three-dimensional trees would have a pyramidal form, with a two-dimensional tree as its base. There are also trivial local trees (consisting of a single root), and even empty ones. Composite trees can be built from local ones by identifying the root of one local tree with a node in the child tree of another (and adapting the addresses in order to incorporate them into the newly created tree domain). Figure 2 shows examples of local and composite trees for the first four steps of the hierarchy (only some composite trees are labeled, and in the three-dimensional case, only the addresses of nodes that do not appear in the rightmost local tree as well are given, for clarity. ε_d denotes an empty sequence of order d).

Rogers [2] also defines automata for labeled $\mathbb{T}d$ s:

Definition 3 A $\mathbb{T}d$ automaton with finite state set Q and (non-ranked) alphabet Σ is a finite set of triples $\mathcal{A}^d \subseteq \Sigma \times Q \times \mathbb{T}_Q^{d-1}$.

The interpretation of a triple $\langle \sigma, q, T \rangle \in \mathcal{A}^d$ is that if a node of a $\mathbb{T}d$ is labeled with σ and T encodes the assignment of states to its children, then that node may be assigned state q . A run of a $\mathbb{T}d$ automaton on a Σ -labeled $\mathbb{T}d$ $T = \langle T, \tau \rangle$ is a mapping $r : T \rightarrow Q$ in which each assignment is licensed by \mathcal{A}^d . Note that this implies that a leaf labeled with σ may be assigned state q only if there is a triple $\langle \sigma, q, \emptyset \rangle \in \mathcal{A}^d$, where \emptyset is the empty $\mathbb{T}(d-1)$. If $F \subseteq Q$ is the set of accepting states, then the set of (finite) Σ -labeled $\mathbb{T}d$ recognized by \mathcal{A}^d is that set for which there is a run that assigns the root a state in F .

$\mathbb{T}1$ automata correspond to finite-state automata for strings, i.e., they recognize the regular languages. $\mathbb{T}2$ automata correspond to (non-deterministic) finite-state automata for trees, i.e., they recognize the regular tree languages.

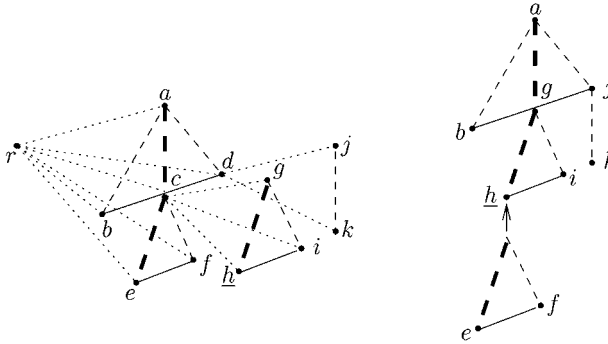


Figure 3. Ambiguity in the yield for $d \geq 3$, resolved by marked foot nodes

One of the most important concepts in connection with multi-dimensional trees is that of the *yield* of a tree. The yield of a two-dimensional tree is the string formed by its leaf labels. In Rogers' [2] words, it is a projection of the tree onto the next lower level, i.e., its dimensions are reduced by one. Tds with $d \geq 3$ have several yields, one for each dimension that is taken away, down to the one-dimensional string yield. Note that when taking the yield of a tree with $d \geq 3$, some thought has to go into the question of how to interweave the child trees of its local components to form a coherent $(d-1)$ -dimensional tree, since there are often several possibilities. Rogers solves this by introducing special nodes called *heads* and defines them such that in the child tree of every local component there is a unique path of heads leading from the root to a leaf. This leaf is called the *foot* of the child tree and marks the splicing point, i.e., the point where the yield of the subtree containing it should be connected to the remaining part of the overall yield. See [2] for the exact definition.

As is well known, the class of the string yields of languages recognized by (two-dimensional) finite-state tree automata are the context-free languages. The class of the string yields of d -dimensional tree languages for $d \geq 3$ are situated between the classes of context-free and context-sensitive languages in the Chomsky Hierarchy, where for every d the class of string yields of the d -dimensional tree languages is properly contained in the next (i.e., for $d+1$).

Via the yield operation, Rogers has established a link between T3s and TAGs by proving the equivalence of T3 recognizing automata and non-strict TAGs:

Theorem 4 ([2]) *A set of Σ -labeled two-dimensional trees is the yield of a recognizable set of Σ -labeled T3 iff it is generated by a non-strict TAG.*

The representation of a TAG as three-dimensional trees obviously constitutes a regularization: Trees are now constructed by adding local trees at the frontier of another tree (see Figure 4), which is a regular process, instead of expanding nodes at the interior. As follows from Theorem 4, the trees generated by the original TAG can be extracted from the T3s using the yield operation.

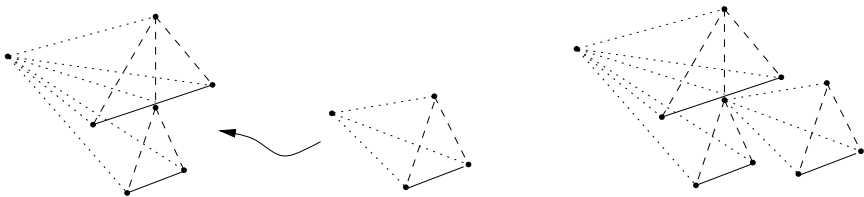


Figure 4. Adjunction in TAG expressed via three-dimensional trees

Rogers conjectures that there may also be potential linguistic applications for structures of more than three dimensions, and gives an amelioration of the standard TAG account of modifiers using four dimensions (see [1]).

In the next section we will introduce a new notation for multi-dimensional trees that is a generalization of the one on which (classical) finite-state tree automata are based, i.e., a representation that allows multi-dimensional trees to be noted as expressions over a partitioned alphabet.

3. Multi-dimensional Trees as Terms

We will use finite d -dimensional tree labeling alphabets Σ^d where each symbol $f \in \Sigma^d$ is associated with at least one unlabeled $(d - 1)$ -dimensional tree t specifying the admissible child structure for a root labeled with f (as before it is possible to admit several, albeit finitely many, child structure trees for one symbol). To all intents and purposes t can be given in any form suitable for trees, as long as it is compatible with the existence of an empty tree, but for consistency's sake we will use the definition of multi-dimensional trees given below from here on and write t as an expression over a special kind of “alphabet” containing just one symbol ρ for which any child structure is admissible.

Let Σ_t^d for $d \geq 1$ be the set of all symbols associated with t and Σ^0 a set of constant symbols. The set \mathbb{T}_{Σ^d} of all d -dimensional trees can then be defined inductively as follows:

Definition 5 *Let ε^d be the empty d -dimensional tree. Then*

- $\mathbb{T}_{\Sigma^0} := \{\varepsilon^0\} \cup \Sigma^0$, and
- for $d \geq 1$: \mathbb{T}_{Σ^d} is the smallest set such that $\varepsilon^d \in \mathbb{T}_{\Sigma^d}$ and $f[t_1, \dots, t_n]_t \in \mathbb{T}_{\Sigma^d}$ for every $f \in \Sigma_t^d$, $t \in \mathbb{T}_{\{\rho\}^{d-1}}$, n the number of nodes in t , $t_1, \dots, t_n \in \mathbb{T}_{\Sigma^d}$ and t_1, \dots, t_n are rooted breadth-first in that order¹ at the nodes of t .

Note how naturally this generalization comprises the concept of rank for labeling symbols in two-dimensional trees: For every symbol f in Σ_s^2 for some string s , s encodes the rank of f in its length, and specifies that the children of a node labeled with f should be ordered linearly (as is normal in two-dimensional trees). For $d \geq 3$ the term “rank” of some symbol $g \in \Sigma_t^d$ still makes sense if we indicate the number of nodes in t by it –

¹This is an ad hoc settlement, any other spatial arrangement would do as well.

this way most of the results for two-dimensional trees can be read directly as applying to multi-dimensional trees in general.

For some tree $t_p = f[t_1, \dots, t_n]_t$ with $f \in \Sigma_t^d$, t_1, \dots, t_n are the direct subtrees of the tree, and the rest of the usual tree terminology can be applied in a similar manner. Also, for some fixed d , let \square be a special symbol associated with ε^{d-1} (leaf label). A tree $c \in \mathbb{T}_{\Sigma^d \cup \{\square\}}$ in which \square occurs exactly once is still called a context, and $c[[s]]$ for $c \in C_{\Sigma^d}$ and $s \in \mathbb{T}_{\Sigma^d}$ is defined via substitution as before.

Our new notation is equivalent to the one by Rogers in the following sense: For every recognizable set $L_R \subseteq \mathbb{T}_{\Sigma}^d$ of d -dimensional trees over some alphabet Σ in Rogers' notation there is a translation $\Phi : L_R \longrightarrow \mathbb{T}_{\Sigma^d}$ characterized by:

- For $d = 0$: $\langle \emptyset, \emptyset \rangle \mapsto \varepsilon^0$ and, for some $a \in \Sigma$, $\langle \{1\}, \{1 \mapsto a\} \rangle \mapsto a$.
- For $d \geq 1$: $\langle \emptyset, \emptyset \rangle \mapsto \varepsilon^d$, $\langle \{\langle \rangle\}, \{\langle \rangle \mapsto a\} \rangle \mapsto a$ for some $a \in \Sigma$, and, for some $f \in \Sigma$, $\langle \{\langle \rangle\} \cup T_x, \{\langle \rangle \mapsto f\} \cup \tau_x \rangle \mapsto f[\Phi(\langle T_1, \tau_1 \rangle), \dots, \Phi(\langle T_n, \tau_n \rangle)]_t$ with $t = \Phi(\langle T_t, \tau_t \rangle)$ where T_t is the set of first elements of the members of T_x and τ_t is the unique function $\tau_t : T_t \longrightarrow \{\rho\}$, and $T_i = \{z | \langle a_i \rangle \cdot z \in T_x\}$ for $1 \leq i \leq n$ where a_i is the i th element in the sequence obtained by ordering the members of T_t inductively by length. τ_i is defined by $\tau_i(z) = \tau_x(\langle a_i \rangle \cdot z)$.

Σ^d is obtained as follows: For each term $t_p \in \Phi(L_R)$ and each subterm $f[t_1, \dots, t_n]_t$ of t_p , $f \in \Sigma_t^d$. We have restricted ourselves to recognizable sets of trees (i.e., that are built from a finite set of local trees) because otherwise Σ^d may be infinite, which is due to the fact that Rogers uses non-partitioned labeling alphabets so that in theory arbitrarily many roots labeled with the same symbol can have completely different child structures.

For every set $L_N \subseteq \mathbb{T}_{\Sigma^d}$ of d -dimensional trees in the notation given above there is a translation $\Psi : L_N \longrightarrow \mathbb{T}_{\Sigma}^d$ characterized by the following (the construction of Σ from Σ^d is trivial):

- For $d = 0$: $\varepsilon^0 \mapsto \langle \emptyset, \emptyset \rangle$ and, for $a \in \Sigma^0$, $a \mapsto \langle \{1\}, \{1 \mapsto a\} \rangle$.
- For $d \geq 1$: $\varepsilon^d \mapsto \langle \emptyset, \emptyset \rangle$, $a \mapsto \langle \{\langle \rangle\}, \{\langle \rangle \mapsto a\} \rangle$ for some $a \in \Sigma_{\varepsilon^{d-1}}^d$, and, for some $f \in \Sigma_s^d$, $f[s_1, \dots, s_m]_s \mapsto \langle \{\langle \rangle\} \cup T_y, \{\langle \rangle \mapsto f\} \cup \tau_y \rangle$ where $T_y = \bigcup_{1 \leq i \leq m, x \in S_i} \langle b_i \rangle \cdot x$ with $\langle S_i, \sigma_i \rangle = \Psi(s_i)$ for all i with $1 \leq i \leq m$ and $\langle S_s, \sigma_s \rangle = \Psi(s)$ and b_i is the i th element in the sequence obtained by ordering the elements of S_s inductively by length. τ_y is defined by $\tau_y(\langle b_i \rangle \cdot z) = \sigma_i(z)$.

Both translations traverse the input structure recursively, which includes, for every symbol, a recursion through the tree specifying the admissible child structure for that symbol, which in turn entails recursions through the dimensions down to zero (as the child structure tree is translated, too).

We will show the equivalence of the two notations by proving that $\Psi(\Phi(t_p)) = t_p$ for all $t_p \in L_1$ for some arbitrary recognizable $L_1 \subseteq \mathbb{T}_{\Sigma}^d$ and $\Phi(\Psi(t_q)) = t_q$ for all $t_q \in L_2$ for some arbitrary $L_2 \subseteq \mathbb{T}_{\Sigma^d}$ for corresponding alphabets Σ and Σ^d .

1. $\Psi(\Phi(t_p)) = t_p$ for all $t_p \in L_1$: For $d = 0$ this is clear. We will prove the claim for $d \geq 1$ by induction on the depth of t_p . For depth 0 ($t_p = \langle \emptyset, \emptyset \rangle$) and 1 ($t_p = a$ for some $a \in \Sigma$) this is also clear. Assume that the claim holds for all $d_1 < d$ and all d -dimensional trees with depth k for some $k \geq 0$. Assume that $t_p = \langle \{\langle \rangle\} \cup T_x, \{\langle \rangle \mapsto f\} \cup \tau_x \rangle$ for some $f \in \Sigma$ has depth $k + 1$.

$$t_p = \langle \{ \langle \rangle \} \cup \{ \langle a_1 \rangle \} \cdot T_1 \cup \dots \cup \{ \langle a_n \rangle \} \cdot T_n, \{ \langle \rangle \mapsto f \} \cup \bigcup_{z \in T_1} (\langle a_1 \rangle \cdot z \mapsto \tau_1(z)) \cup \dots \cup \bigcup_{z \in T_n} (\langle a_n \rangle \cdot z \mapsto \tau_n(z)) \rangle$$

(definition of T_x and τ_x , with T_i, τ_i, a_i defined as above)

$$\Psi(\Phi(t_p)) = \langle \{ \langle \rangle \} \cup \{ \langle b_1 \rangle \} \cdot S_1 \cup \dots \cup \{ \langle b_m \rangle \} \cdot S_m, \{ \langle \rangle \mapsto f \} \cup \bigcup_{z \in S_1} (\langle b_1 \rangle \cdot z \mapsto \sigma_1(z)) \cup \dots \cup \bigcup_{z \in S_m} (\langle b_m \rangle \cdot z \mapsto \sigma_m(z)) \rangle$$

(definition of T_y and τ_y , with S_i, σ_i, b_i defined as above)

$$\Psi(\Phi(t_p)) = \Psi(f[\Phi(\langle T_1, \tau_1 \rangle), \dots, \Phi(\langle T_n, \tau_n \rangle)]_{\Phi(\langle T_t, \tau_t \rangle)}) \text{ (definition of } \Phi \text{).}$$

By the induction hypothesis we know that $\langle S_s, \sigma_s \rangle = \Psi(s) = \Psi(\Phi(\langle T_t, \tau_t \rangle)) = \langle T_t, \tau_t \rangle$ and consequently $n = m$ and $a_i = b_i$ for all $1 \leq i \leq n, m$. In the same way we know that $\langle S_i, \sigma_i \rangle = \Psi(\Phi(\langle T_i, \tau_i \rangle)) = \langle T_i, \tau_i \rangle$ for all i with $1 \leq i \leq n, m$, and thus $\Psi(\Phi(t_p)) = t_p$. ■

2. $\Phi(\Psi(t_q)) = t_q$ for all $t_q \in L_2$: Again, for $d = 0$ this is clear. We will prove the claim for $d \geq 1$ by induction on the depth of t_q . For depth 0 ($t_q = \varepsilon^d$) and 1 ($t_q = a$ for some $a \in \Sigma_{\varepsilon^{d-1}}^d$) this is also clear. Assume that the claim holds for all $d_1 < d$ and all d -dimensional trees with depth k for some $k \geq 0$. Assume that $t_q = f[s_1, \dots, s_m]_s$ for some $f \in \Sigma_s^d$ has depth $k + 1$.

$$\Phi(\Psi(t_q)) = \Phi(\langle \{ \langle \rangle \} \cup \{ \langle b_1 \rangle \} \cdot S_1 \cup \dots \cup \{ \langle b_m \rangle \} \cdot S_m, \{ \langle \rangle \mapsto f \} \cup \bigcup_{z \in S_1} (\langle b_1 \rangle \cdot z \mapsto \sigma_1(z)) \cup \dots \cup \bigcup_{z \in S_m} (\langle b_m \rangle \cdot z \mapsto \sigma_m(z)) \rangle)$$

(definition of T_y and τ_y , with S_i, σ_i, b_i defined as above)

$$= f[\Phi(\langle T_1, \tau_1 \rangle), \dots, \Phi(\langle T_n, \tau_n \rangle)]_{\Phi(\langle T_t, \tau_t \rangle)} \text{ (definition of } \Phi \text{).}$$

We know, by the relevant definitions, that $\langle T_t, \tau_t \rangle = \langle \{ b_1, \dots, b_m \}, \{ b_1 \mapsto \rho, \dots, b_m \mapsto \rho \} \rangle = \Psi(s)$ and thus $\Phi(\langle T_t, \tau_t \rangle) = \Phi(\Psi(s)) = s$ by the induction hypothesis, which also implies $m = n$. By the same reflection, $\langle T_i, \tau_i \rangle = \langle \{ z | \langle b_i \rangle \cdot z \in \{ \langle b_1 \rangle \} \cdot S_1 \cup \dots \cup \{ \langle b_m \rangle \} \cdot S_m \}, \bigcup_{z \in S_i} (\langle b_i \rangle \cdot z \mapsto \sigma_i(z)) \rangle = \Psi(s_i)$ and $\Phi(\langle T_i, \tau_i \rangle) = \Phi(\Psi(s_i)) = s_i$ for all i with $1 \leq i \leq n, m$. This concludes the proof. ■

We can now represent automata for multi-dimensional trees as a direct generalization of classical finite-state tree automata:

Definition 6 A (total, deterministic) finite-state d -dimensional tree automaton is a 4-tuple $\mathcal{A}^d = (\Sigma^d, Q, \delta, F)$ with input alphabet Σ^d , finite set of states Q , set of accepting states $F \subseteq Q$ and transition function δ with $\delta(t(q_1, \dots, q_n), f) \in Q$ for every $f \in \Sigma_t^d$, $t \in \mathbb{T}_{\{\rho\}^{d-1}}$, where $t(q_1, \dots, q_n)$ encodes the assignment of states to the nodes of t ($t(q_1, \dots, q_n)$ is isomorphic to t and its nodes are labeled with q_1, \dots, q_n breadth-first in that order if Q is taken as a partitioned alphabet associating every symbol with all the child structures it occurs with in δ). The function δ extends to d -dimensional trees: $\delta : \mathbb{T}_{\Sigma^d} \longrightarrow Q$ is defined such that if $t_p = f[t_1, \dots, t_n]_t \in \mathbb{T}_{\Sigma^d}$ for $t \in \mathbb{T}_{\{\rho\}^{d-1}}$ then

$\delta(t_p) = \delta(t(\delta(t_1), \dots, \delta(t_n)), f)$. The set of trees accepted by \mathcal{A}^d is $L(\mathcal{A}^d) = \{t_p \in \mathbb{T}_{\Sigma^d} \mid \delta(t_p) \in F\}$.

The equivalence between this definition and the one by Rogers [2] is easy to see: For two corresponding automata $\mathcal{A}^d = (\Sigma^d, Q, \delta, F)$ and $\mathcal{A}_R^d \subseteq \Sigma_R \times Q_R \times \mathbb{T}_{Q_R}^{d-1}$ with the set of accepting states F_R in the two notations the set of states Q and Q_R and accepting states F and F_R coincide, the construction of Σ_R from Σ^d is trivial, and Σ^d is constructed from \mathcal{A}_R^d as follows: $f \in \Sigma_t^d$ for all triples $\langle f, q, t_0 \rangle \in \mathcal{A}_R^d$, where $t = \Phi(\langle T_0, \tau_{0+} \rangle)$ for $t_0 = \langle T_0, \tau_0 \rangle$ and τ_{0+} is the unique function $\tau_{0+} : T_0 \longrightarrow \{\rho\}$. Most importantly, there is a one-to-one correspondence between the elements of \mathcal{A}_R^d and δ : Every triple $\langle f, q, t_0 \rangle \in \mathcal{A}_R^d$ can be translated to an assignment $\delta(\Psi(t_0), f) = q$ of \mathcal{A}^d , and every assignment $\delta(t(q_1, \dots, q_n), f) = q$ of \mathcal{A}^d to a triple $\langle f, q, \Phi(t(q_1, \dots, q_n)) \rangle \in \mathcal{A}_R^d$. From this and from the identical state sets it follows that $L(\mathcal{A}_R^d) = \Psi(L(\mathcal{A}^d))$ and $L(\mathcal{A}^d) = \Phi(L(\mathcal{A}_R^d))$.

With the term representation and the adapted definitions of contexts and automata given in this section, results pertaining to the class of regular string or tree languages as for instance the Myhill-Nerode theorem or the Pumping lemma (see Section 1) and all their consequences (like the existence of a unique minimal finite-state automaton \mathcal{A}_L^d recognizing L for every recognizable d -dimensional tree language L) carry over directly to multi-dimensional trees.

Finally, we will define a yield function for multi-dimensional trees in the new notation. As for $d \geq 3$ the yield is not unambiguous (see Figure 3), the structures have to be enriched with additional information. Assume that, for $d \geq 2$, in every tree $t_p \in \mathbb{T}_{\Sigma^d}$ every labeling symbol $f \in \Sigma^d$ is indexed with a set $S \subseteq \{2, \dots, d\}$. If $x \in S$ then we call a node labeled by f_S a *foot node for the $(x - 1)$ -dimensional yield of t_p* . For every subtree t_q of t_p the distribution of these foot nodes must fulfil certain conditions:

- (1) If t_q has depth 0 the index set in its root label must contain d , otherwise $t_q = f_S[t_1, \dots, t_n]_t$ with $f \in \Sigma_t^d$, $S \subseteq \{2, \dots, d\}$, and $t_1, \dots, t_n \in \mathbb{T}_{\Sigma^d}$ must have exactly one direct subtree $t_i \in \{t_1, \dots, t_n\}$ whose root labeling symbol is indexed with a set containing d and t_i is attached to a *leaf* in t . In both cases, we will refer to that root as the d -dimensional foot node of t_q .
- (2) The foot nodes are distributed in such a way that for every n -dimensional yield of t_p with $n < d$, condition (1) is fulfilled as well.

For $d \geq 2$, the direct yield of a tree $t_p \in \mathbb{T}_{\Sigma^d}$ is then defined recursively as

$$yd_{d-1}(t_p) = \begin{cases} \varepsilon^{d-1} & \text{for } t_p = \varepsilon^d, \\ a_S & \text{for } t_p = a_S \text{ with } a \in \Sigma_{\varepsilon^{d-1}}^d \text{ and } S \subseteq \{2, \dots, d\}, \\ op_{t_p}(t_1) & \text{for } t_p = f_S[t_1, \dots, t_n]_t \text{ with } t_1, \dots, t_n \in \mathbb{T}_{\Sigma^d}, f \in \Sigma_t^d, \\ & t \neq \varepsilon^{d-1}, \text{ and } S \subseteq \{2, \dots, d\}, \end{cases}$$

where $op_{t_p}(t_i)$ for $t_i \in \{t_1, \dots, t_n\}$ is defined as the $(d - 1)$ -dimensional tree that is obtained by replacing the d -dimensional foot node of t_i in $yd_{d-1}(t_i)$ by $e_R[op_{t_p}(t_j), \dots, op_{t_p}(t_k)]_{t_x}$, where e_R with $e \in \Sigma^d$ and $R \subseteq \{2, \dots, d\}$ is the label of the foot node, t_x is the $(d - 2)$ -dimensional child structure of the node at which t_i is attached in t and t_j, \dots, t_k are the direct subtrees of t_p that are attached (breadth-first in that order) at the nodes of t_x .

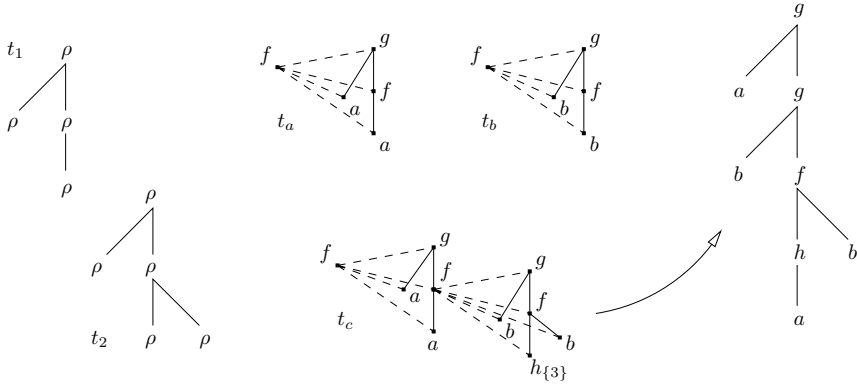


Figure 5. Example 2

The result $yd_{d-1}(t_p)$ is a $(d-1)$ -dimensional tree over an alphabet Σ^{d-1} containing at least all the node labels in $yd_{d-1}(t_p)$, each associated at least with the child structures it occurs with. Obviously, the string yield of a d -dimensional tree for $d \geq 2$ can be obtained by taking the direct yield $d-1$ times.

Example 2 defines an automaton \mathcal{A}_{ww}^3 recognizing a three-dimensional tree language whose set of string yields $yd_1(L(\mathcal{A}_{ww}^3))$ is $L_{ww} = \{ww | w \in \{a, b\}^+\}$:

Example 2 $\mathcal{A}_{ww}^3 = (\Sigma^3, \{q_a, q_b, q_g, q_y, q_z, q_f, q_x\}, \delta, \{q_f\})$ where $\Sigma^3 = \{a, b, f, g, h_{\{3\}}\}$ with $a, b, f, g, h_{\{3\}} \in \Sigma_{\varepsilon^2}^3$ and $f \in \Sigma_{t_1}^3$ for $t_1 = \rho[\rho[\varepsilon^1, \rho[\rho[\varepsilon^0]_{\rho[\rho[\varepsilon^0]_{\rho}}]_{\rho[\rho[\varepsilon^0]_{\rho}}]_{\rho}]]_{\rho}$ and $f \in \Sigma_{t_2}^3$ for $t_2 = \rho[\rho[\varepsilon^1, \rho[\rho[\varepsilon^1, \rho[\rho[\varepsilon^0]_{\rho[\rho[\varepsilon^0]_{\rho}}]_{\rho[\rho[\varepsilon^0]_{\rho}}]_{\rho}]]_{\rho}]]_{\rho}$. (Note that in Σ^3 only index sets containing 3 have been given, as the distribution of foot nodes for the string yield is never ambiguous). δ is defined as follows:

$$\begin{aligned}
 \delta(\varepsilon^2, a) &= q_a & \delta(t_1(q_g, q_a, q_z, q_a)) &= q_f \\
 \delta(\varepsilon^2, b) &= q_b & \delta(t_1(q_g, q_b, q_z, q_b)) &= q_f \\
 \delta(\varepsilon^2, f) &= q_z & \delta(t_2(q_g, q_a, q_z, q_y, q_a)) &= q_z \\
 \delta(\varepsilon^2, g) &= q_g & \delta(t_2(q_g, q_b, q_z, q_y, q_b)) &= q_z \\
 \delta(\varepsilon^2, h_{\{3\}}) &= q_y
 \end{aligned}$$

and $\delta(t_0, x) = q_x$ for all other admissible t_0 and all symbols $x \in \Sigma^3$. Figure 5 shows t_1 and t_2 , three trees $t_a, t_b, t_c \in L(\mathcal{A}_{ww}^3)$ in the middle, and the two-dimensional yield for t_c , whose one-dimensional yield is the string $abab$.

4. Conclusion

We have provided a new, term-like representation for multi-dimensional trees which establishes them as a direct generalization of classical trees. As a consequence multi-dimensional trees can now be used as an input for (slightly adapted) finite-state applications based on classical formal (tree) language theory, for example in the areas of grammatical inference (shown in [10]) or natural language processing. Via the concept of the yield of a multi-dimensional tree this also means that these applications can now be conceived to be able to process even some language classes that lie beyond context-freeness.

Due to lack of space we have not furnished the full possible system of concepts linked to recognizable multi-dimensional tree languages, but of course further notions such as regular grammars can be formulated for these structures as well. Also, various results can be ameliorated such as a less complex-looking, regular version of the Pumping lemma for the string languages generated by TAGs [14] relying on the correspondence to three-dimensional trees (see Section 2).

Another interesting project we propose for the near future would be to check whether any implementations of known finite-state applications based on formal tree languages can be adapted to multi-dimensional trees, or even if with this generalization new implementations have become possible.

References

- [1] J. Rogers. Syntactic structures as multi-dimensional trees. *Research on Language and Computation*, 1:265–305, 2003.
- [2] J. Rogers. wMSO theories as grammar formalisms. *Theoretical Computer Science*, 293:291–320, 2003.
- [3] A.K. Joshi. Tree adjoining grammars: How much context-sensitivity is required to provide reasonable structural description. In D. Dowty, L. Karttunen, and A. Zwicky, editors, *Natural Language Processing*. Cambridge University Press, 1985.
- [4] A. Kasprzik. Two equivalent regularizations of tree adjoining grammars. Technical Report 08-1, University of Trier, 2008. Available on: http://www.mathematik.uni-trier.de/tf/08_01.pdf.
- [5] F. Morawietz. Two-step approaches to natural language formalisms. *Studies in Generative Grammar*, 64, 2003.
- [6] D.J. Weir. A geometric hierarchy beyond context-free languages. *Theoretical Computer Science*, 104(2):235–261, 1992.
- [7] D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987.
- [8] Y. Sakakibara. Learning context-free grammars from structural data in polynomial time. *Theoretical Computer Science*, 76(2–3):223–242, 1990.
- [9] F. Drewes and J. Högberg. Learning a regular tree language from a teacher. In Z. Ésik and Z. Fülöp, editors, *Developments in Language Theory 2003*, volume 2710 of *LNCS*, pages 279–291. Springer, 2003.
- [10] A. Kasprzik. A learning algorithm for multi-dimensional trees, or: Learning beyond context-freeness. Technical Report 08-2, University of Trier, 2008. Available on: www.mathematik.uni-trier.de/tf/08_02.pdf.
- [11] J.E. Hopcroft and J.D. Ullman. *Introduction to automata theory, languages, and computation*. Addison-Wesley, 1979.
- [12] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on: <http://www.grappa.univ-lille3.fr/tata>, 2005. release October, 12th 2007.
- [13] F. Gécseg and M. Steinby. Tree languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, chapter 1, pages 1–68. Springer, 1997.
- [14] K. Vijayashanker. *A Study of Tree-Adjoining Grammars*. PhD thesis, University of Pennsylvania, 1987.

Transducer Minimization and Information Compression for NooJ Dictionaries

Slim MESFAR and Max SILBERZTEIN

LASELDI, Université de Franche-Comté, Besançon, France

Abstract. In this paper, we describe the use of an incremental construction method of minimal, acyclic, deterministic FST. The approach consists in constructing a transducer in a single step by adding new strings one by one and minimizing the resultant automaton incrementally. Then, we present a new method to encode the morphological information associated with the dictionary entries. The new encoding unifies a large number of word forms' analyses, thus reducing the number of terminal states of the dictionary's FST, that triggers a more efficient minimization process. Finally, we present experimental results on the FST that represents the Arabic dictionary.

Keywords. Automata compression, sequential minimization, electronic dictionaries, NooJ

Introduction

Finite-State automata (FSA) are used in a variety of Natural Language Processing (NLP) applications. In particular, they provide efficient storage and retrieval of finite sets of strings over a finite alphabet, and thus can be used to represent the vocabulary of a language. Finite-State Transducers (FSTs)¹ allow users to associate each element of a vocabulary with some information, such as morpho-syntactic information (POS - Part Of Speech, Gender, Number, etc.), syntactic and semantic information (e.g. transitive, Human, etc.), or more complex, such as the term's translation in other languages, a set of synonymous expressions, etc. The purpose of a lexical analysis of a text is to map tokens that occur in texts into a vocabulary, usually described in a dictionary. In most languages, tokens are inflected and/or derived word forms that need to be associated with the corresponding lexeme (dictionary entry) and some linguistic information. Thus, transducers are well adapted to perform automatic lexical analyses. FSA can be easily extended into FSTs to produce information associated with the accepted tokens: the output of the transducer is simply affixed to the corresponding accepting states² of the FSA. But then, special encoding techniques are necessary to represent the sets of information produced by the FST. Since these transducers can be represented very efficiently and are associated with linear lookup methods, we are using them to represent dictionaries within the lin-

¹In automata theory, they are also called Mealy automata [1].

²An accepting state (also called terminal state) is the final state of a recognized word form. It's usually represented by a double circle (see Figure 1).

guistic platform NooJ³. Especially, deterministic acyclic ones (i.e. Deterministic Acyclic Finite State Transducers = DAFST) are of particular interest for NLP applications. DAFSTs representing dictionaries can be constructed in various ways; works presented in [2], [3], [4] and [5] present a good reference of the algorithms used to process vocabularies with FSA.

1. Related Work

In this section, we present a chronological taxonomy of the various algorithms for deterministic acyclic finite state automata construction. This taxonomy is, partially, described in [1] and [6]:

- In the early-1990s, Revuz derived the first linear DAFSA⁴ minimization algorithms [7] and [8]. The primary algorithm presented by Revuz uses an ordering of the words to quickly compress the endings of the words within the dictionary.
- By the mid-1990s, several groups were working independently on incremental algorithms - most of which are the same or very similar.
- In 1996, B. W. and R. E. Watson derived an incremental algorithm. Unlike many of the other derivations of related algorithms, their algorithm provides facilities for removing words from the language accepted by the automaton, while maintaining minimality.
- Also in 1996 - 1997, Daciuk independently derived the generalized incremental algorithm. In addition, Daciuk derived a new incremental algorithm which adds the words in lexicographic order. Simultaneously, Mihov had also derived the sorted algorithm. Daciuk and Mihov went on to publish the algorithms in their dissertations as [2] and [9], respectively.
- Independently in 1997, in the field of verification, Holzmann and Puri [10] discovered a restricted form of the algorithm, in which all words accepted by the automaton have the same length.
- In 1998, B. W. Watson sketched the semi-incremental algorithm in [11].
- In 2000, Revuz presented essentially the generalized algorithm [12]; though he also sketched word deletion algorithms similar to those previously derived by B. Watson and R. Watson.
- In 2001, Graña et al. summarized some of the current results and made improvements to several of the algorithms [3].
- Next, in 2002, the generalized algorithm was straightforwardly extended by Carasco and Forcada to handle cyclic automata [13].
- In [14], B. W. Watson gave a fast and simple incremental algorithm based upon Brzozowski's minimization algorithm.
- Recently, in 2007, L. Tounsi and al. [15] noticed that even using an incremental construction, it was not enough to have a reduced automaton to represent electronic dictionaries. They proposed the introduction of a compression method for

³NooJ is a linguistic development environment that can be used to build large-coverage formalization of languages (from the alphabetical to the semantic level), as well as a corpus processor that can apply sophisticated queries to texts (e.g. look for noun phrases followed by a date). NooJ is freeware and can be downloaded from <http://www.nooj4nlp.net>.

⁴DAFSA : Deterministic Acyclic Finite State Automata.

minimized automata. To reduce the memory used by an automaton, a search for repetitive structures, sub-automata, was performed.

The first NooJ engine used a traditional method to obtain a minimal transducer by first creating a deterministic (not minimal) transducer for the dictionary and then minimizing it using an efficient algorithm. The first stage was usually performed by building a trie, for which linear-time minimization algorithms are associated. Transducer minimization algorithms are quite efficient in terms of the size of their input (the trie): NooJ's algorithm used memory and time requirements that are linear in terms of the number of states in the input trie. Unfortunately, even such linear performance is not sufficient when the trie is much larger than the available physical memory. For instance, the Hungarian dictionary, which contains over 50,000 entries, produces a 130+ million word form trie that contained over 100 million states (each state requires 10 bytes in average). Some effort towards decreasing the memory requirement has been made; such as those done by Revuz [16] and Daciuk [8]. This paper presents a way to reduce these intermediate memory requirements and decrease the total construction time by constructing the minimal dictionary incrementally (word by word, maintaining an invariant of minimality), thus avoiding ever having to construct the trie in memory.

2. Incremental construction of NooJ automata

The Myhill-Nerode theorem [1] states that among the many deterministic automata that accept a given language, there is a unique automaton (up to isomorphism) that has a minimal number of states. This is called the minimal deterministic automaton of the language. An incremental minimization of a DAFSA means that every time a word is added to the automaton, it is minimized. This could lead to modifications of the parts of the automaton that had previously been minimized. A list of operations has to be done every time a word is added:

- Search for the common prefix in the automaton.
- Follow the common prefix unless confluence states⁵ encountered.
- Remove the last state from the Register⁶ R.
- Clone⁷ all states in the common prefix from the first confluence state.
- Create path for the suffix, mark the final state.
- Starting from the final state towards the start state, either put states from the path in R, or replace them with equivalent ones. If preceding state is in R, remove it from R. Stop when no more changes occur.

We notice that:

- If there are no confluence states in the common prefix, we have to simply append the word suffix to the last common state.
- If there are confluence states in the common prefix, appending another transition to the last state in the path would accidentally add more words than desired (see Figure 1).

⁵A confluence state is a state that is target of more than one incoming transition.

⁶Register R : the set of states with unique right language \overline{L} (cf. definition in the 8th footnote).

⁷Cloning a state is creating a new state that has outgoing transitions with the same labels and to the same destinations states as the given one.

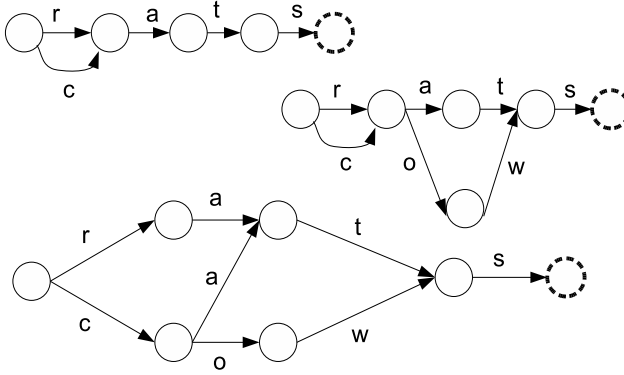


Figure 1. The result of blindly adding the word cows to a minimized dictionary (appearing to the left) containing rats and cats. The rightmost dictionary accidentally contains *rows* as well. The lower dictionary is correct; state 2 had to be cloned.

- The previous states in the prefix path may need to be changed, because the right languages⁸ \vec{L} of those states may have changed; we must recalculate the equivalence relation for all states on the path of the new word.
- To check if $q \equiv q'$ (q is equivalent to q'), we only check if both are final or non-final, and if their suites of transitions are identical (the same number, labels and targets).

Minimizing the automaton, on-the-fly, could change the equivalence classes of some states each time a word is added. Before constructing a new state in the dictionary, we first determine if it would be included in the equivalence class of a pre-existing state. In addition, we might need to change the equivalence classes of previously constructed states since their languages might have changed. This leads to an incremental construction algorithm. In the Figure 1, we develop the example discussed in [17].

However, processing complex queries with NooJ requires complex information to be associated with each word of a vocabulary [18]. If we wish to perform a syntactic parsing of each sentence of a text, we need to associate each word with its lemma (base form or stem), its syntactic category (Noun, Verb, etc.), its morphological attributes (Number, Gender, Tense, etc.), syntactic and semantic information such as a verb's number and type of complements, a predicative noun's list of support verb, translations, etc. NooJ's linguistic units are recognized by looking up dictionaries, using morphological grammars to parse word forms as well as using syntactic grammars to parse phrases. Each of the recognized/matching sequence of the text is then associated with a complex string of information codes: its lemma, its syntactic category (e.g. Noun), inflectional codes (e.g. masculine, plural, Past Participle, etc.), syntactic codes (e.g. "transitive verb"), distributional codes (e.g. "Human noun") and semantic domains (e.g. "medical term").

For instance, here are three typical analyses of three word forms:

⁸The right language of a state q , $\vec{L}(q)$, is the set of all strings, over the alphabet, on a path starting from the state q and reaching any final state of the automaton.

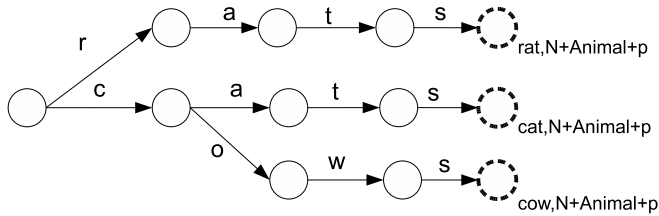


Figure 2. The result of adding the three words (*rats*, *cats* and *cows*) with the associated set of information. The resulting FSA looks like a trie.

- **rats:** rat,N+Animal+p
- **cats:** cat,N+Animal+p
- **cows:** cow,N+Animal+p

Note that these three word forms are associated with the same information string (Noun, Animal, Plural) but their lemmas are different: "rat" for "rats", "cat" for "cats" and "cow" for "cows". Adding these word forms, associated with their information, in an automaton could generate the following FSA that looks like a trie since these different analyses would lead to different transitions (see Figure 2). As we add more and more precise information for each dictionary's entry, even using an incremental construction algorithm, the required compilation memory grows, the FSA's minimization is less and less efficient, and the resulting FSA looks more and more like a trie (see Figure 2), which makes the size of the resulting dictionary too big to be processed by current PCs especially for certain languages such as Arabic or Hungarian. Therefore, we need to explore new approaches to ensure that the minimization can fully be applied, even as the information becomes more and more precise. In order to unify these three analyses, we replace each lemma with a morphological command based on use of the deletion operator with the special code $\langle B \rangle$ (for "Backspace") that computes it from the word form. In other words, in order to compute the lemma "cat" from the word form "cats", we delete the last letter of the entry which corresponds to the command " $\langle B \rangle$ ". This encoding allows us to associate the three word forms "cats", "rats" and "cows" with a unique analysis:

- **rats:** $\langle B \rangle$,N+Animal+p
- **cats:** $\langle B \rangle$,N+Animal+p
- **cows:** $\langle B \rangle$,N+Animal+p

Thanks to this encoding, a large number of word forms can be associated with the same exact analysis⁹. As all analyses are stored in a hash table, the size of the resulting hash table is reduced significantly, and, more importantly, a large number of input strings such as "cats" and "cows" lead to a small number of common terminal states, that triggers a very efficient minimization process (see Figure 3).

⁹More complex inflectional paradigms can be processed with this method:

- helped, suffered, turned ... could be analyzed as $\langle B2 \rangle$: delete 2 letters from the inflected form to get the lemma: turned \Rightarrow turn ;
- men, women could be analyzed as $\langle B2 \rangle$ an : delete the two last letters from the inflected form, and then add "an", e.g. women \Rightarrow wom \Rightarrow woman.

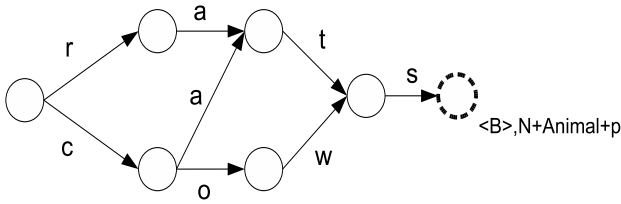


Figure 3. The result of adding the three words (*rats*, *cats* and *cows*) using a compression function for related set of information. Note that the suffix "s" produces a single information string for added word forms.

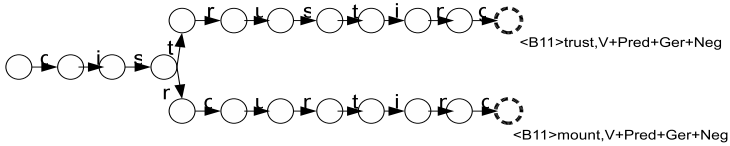


Figure 4. The result of adding the two words (*distrusting*, *dismounting*) with the associated set of information. The resulting FST is a trie.

This solution works very well when the difference between word forms and their lemma is located only in their suffix, which is the case for English and Romance Languages. NooJ can also be used to formalize prefixations. For instance, a NooJ dictionary can be used to produce the negative gerundive form (Gerundive+Neg) of a predicative verb (V+Pred) such as the following analyses:

- **distrusting:** trust,V+Pred+Gerundive+Neg
- **dismounting:** mount,V+Pred+Gerundive+Neg

If we use the previous algorithm to compute the lexeme (e.g. trust) from the word form (distrusting), we get the following encoding:

- **distrusting:** $\langle B11 \rangle$ trust,V+Pred+Gerundive+Neg
- **dismounting:** $\langle B11 \rangle$ mount,V+Pred+Gerundive+Neg

In consequence, the two word forms "distrusting" and "dismounting" (as well as a large number of word forms, such as "disrespecting", "disregarding", etc.) will have to be analyzed differently, even though their analyses are very similar (see Figure 4). This encoding uses the $\langle B \rangle$ operator (delete last letter) to compute the common prefix of the word form and the lemma. Then, it concatenates the resulting prefix with the remaining suffix of the lemma. For instance, to link the word form "had" to its lemma "have", it computes the command " $\langle B \rangle$ ve": delete the last letter of the word form, then add the suffix "ve". This algorithm is well adapted to languages for which inflectional and derivational morphology is performed by adding suffixes to lemmas, such as English and Romance languages. However, it produces poor results for other languages, such as Dutch and German (see for instance in [2] how J. Daciuk needed to add two operators to deal with specific cases in these languages). For Semitic languages such as Arabic and Hebrew, where basic inflectional and derivational rules modify prefixes and infixes massively, we need a more generalized approach, (see Section 5). As a consequence,

Forms	Analysis	Morphological operations
distrusting, dismounting	$\langle B3 \rangle \langle LW \rangle \langle S3 \rangle$	Delete the 3 last characters, move to beg. of word, then delete 3 next letters

Table 1. A unified morphological analysis

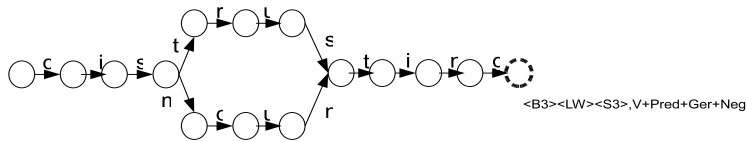


Figure 5. Prefix in a deterministic minimal FSA.

NooJ’s dictionaries for these languages were represented by FSTs that were largely tries: the standard minimization process could not do much to prevent the FST to grow out of control. We needed a new method to link word forms to their lemmas.

3. A new encoding routine

NooJ formalizes inflectional and derivational paradigms by means of Finite-State Transducers (FSTs), entered either via NooJ’s FST graphical editor, or via two-tape regular expressions. FSTs’ inputs are strings of letters and morphological operators; their outputs represent the linguistic analysis of the word forms that are produced. NooJ provides a dozen of morphological operators other than $\langle B \rangle$, including the following ones:

- $\langle L \rangle$: move left,
- $\langle R \rangle$: move right,
- $\langle D \rangle$: duplicate current letter,
- $\langle S \rangle$: remove accent on the current letter,
- etc.

NooJ’s morphological engine uses these operators to perform transformations inside strings. They can be associated with two argument types; either a number (e.g. $\langle L3 \rangle$: go left 3 times) or a "W" (e.g. $\langle LW \rangle$: go to beginning of word). These commands operate on a stack: each of them runs in constant time. Thus, they can link a lemma to its corresponding forms in linear time.

We propose a new encoding routine that computes a series of NooJ’s morphological operators (and no longer only $\langle B \rangle$) to automatically encode each word form’s analysis.

The word forms of Table 1 are then analyzed in a unified way. The resulting FST is represented in Figure 5.

The new analysis can be shared by many word forms, hence these word forms lead to a common terminal state in the dictionary’s FST, which in turn triggers a very efficient minimization process. Moreover, the hash table that stores all different analyses is reduced in size. The algorithm key is computing the series of operators that computes the lemma from each word form, and essentially "compressing" it. In order to compute the new analysis automatically, we recursively compute all the common affixes between the

word form and its lemma. The resulting encoding string uses the four morphological operators $\langle L \rangle$, $\langle R \rangle$, $\langle S \rangle$ and $\langle B \rangle$. This method is described in Algorithm **CompressLemma** below:

Algorithm 1 CompressLemma

Input: string Entry, string Lemma

Output: string Transform

```

if Entry is a prefix of Lemma then
  //mange, manger  $\Rightarrow$  "r";
  return Lemma's prefix;
else if Lemma is a prefix of Entry then
  //mangerons, manger  $\Rightarrow \langle B3 \rangle$ ;
  return " $\langle B \rangle + (\text{length of } \text{Entry}'s \text{ suffix}) + "$ ";
else
  return RecCompression(" $\langle LW \rangle$ ", Entry, Lemma) ;
  // cf. Appendix A; disrespecting, respect  $\Rightarrow "$   $\langle B3 \rangle \langle LW \rangle \langle S3 \rangle$ ";
end if

```

The CompressLemma method locates recursively the longest common substring between the word form and its corresponding lemma¹⁰. It uses a dynamic programming technique¹¹, with a complexity of $\mathcal{O}(n \cdot m)$ where n and m are respectively the word form and lemma lengths; therefore the main algorithm runs in $\mathcal{O}(n \cdot m \cdot \log(n))$. The dynamic programming method used within the **LongestCommonSubstring (LCS)** function is used to avoid traditional solving routines checking, for each of the m starting points of *Lemma*, for the longest common string starting at each of the n starting points of *entry*; in that case, the checks could attend a total of $\mathcal{O}(m^2 \cdot n)$ time. Within the LongestCommonSubstring method, we first find the longest common suffix for all pairs of prefixes of both strings. We define $L_{i,j}$ a matrix containing the maximum length of common strings that end at $Lemma[i]$ and $Entry[j]$.

Next,

if $Lemma[i] = Entry[j]$ **then**

$$L_{i,j} := 1 + L_{i-1,j-1}$$

else

$$L_{i,j} := 0$$

end if

Then, the maximal of these longest common suffixes of possible prefixes must be the longest common substrings of *Lemma* and *Entry*.

$$LCS(Entry, Lemma) := \max_{1 \leq i \leq m, 1 \leq j \leq m} L_{i,j}$$

¹⁰Actually, we simplify the result of CompressLemma if it ends with a delete operator ($\langle S \rangle$). For example, we rewrite " $\langle LW \rangle \langle S3 \rangle \langle R4 \rangle \langle S3 \rangle$ " with " $\langle B3 \rangle \langle LW \rangle \langle S3 \rangle$ "; note that both commands are equivalent.

¹¹Dynamic programming technique was originally introduced in the 1940s by Richard Bellman to describe the process of solving problems exhibiting the properties of overlapping subproblems and optimal substructures that takes much less time than naïve methods.

Dictionary		Verbs	Deverbals	Nouns
Number of word forms		1,290,795	1,443,327	280,267
Number of analyses with:	Original encoding	1,090,516	1,221,277	18,869
	new encoding	53,814	69,007	10,489
Compression rate		95.1%	94.3%	44.4%

Table 2. Experiments on Arabic dictionaries.

This algorithm runs in $\bigcirc(n \cdot m)$ time and memory. To reduce the memory usage of this implementation, we keep only the last and current row of the $L_{i,j}$ table to save memory $\bigcirc(\min(n, m))$ instead of $\bigcirc(n \cdot m)$. The optimized version, including the memory usage reduction is proposed in **Appendix B**.

Although algorithm CompressLemma was designed to solve the prefixation problem, it turns out to be very well adapted to the problem of letter substitutions. For instance, in French, when conjugating the verb "semer", the first vowel "e" is replaced with the accented letter "è" to produce the form "sème". NooJ's original algorithm encodes the analysis of the word form as:

sème [$\langle B3 \rangle$ *emer*] \Rightarrow **semer**

whereas the new algorithm produces the following analysis:

sème [**r** $\langle LW \rangle$ $\langle R \rangle$ $\langle S \rangle$ *e*] \Rightarrow **semer**

While more complex to read, the new analysis unifies a large number of verb conjugation paradigms: (gèle \Rightarrow geler), (lève \Rightarrow lever), (mène \Rightarrow mener), (pèse \Rightarrow peser), (sème \Rightarrow semer), etc. In practice, even dictionaries for Romance languages benefit enormously from the new algorithm.

4. Experiments

The new incremental minimization algorithm and new encoding method are combined together to carry out some performance measurements. While this combination has been successfully used for every language processed in NooJ, the most spectacular results have been obtained with Hungarian: the dictionary that contains 130+ million entries is stored into a 5-million state FST. Arabic makes a heavy use of prefixes: for instance, from the lexical entry "*kataba*" (to write), we get 122 word forms, including the following ones:

- "*áakotubu*" (I write) $\Rightarrow \langle B \rangle \langle LW \rangle \langle S2 \rangle \langle R \rangle a \langle S \rangle \langle R \rangle a \langle S \rangle \langle R \rangle a$
- "*yakotubaāni*" (They write[dual]) $\Rightarrow \langle B3 \rangle \langle LW \rangle \langle S2 \rangle \langle R \rangle a \langle S \rangle \langle R \rangle a \langle S \rangle$
- "*katabā*" (They wrote [dual]) $\Rightarrow \langle B \rangle$

Comparing the old and the new algorithm for Arabic, we get the Table 2. Compiling NooJ's dictionaries with the new algorithms - incremental construction and dynamic information compression - has not taken significantly more time than before. For instance, the full Arabic dictionary (3,014,389 entries) compiles in 4 minutes on a 3.2 Ghz Pentium PC, 2 GB RAM, which is more than adequate for a typical NooJ use (dictionaries are usually recompiled once every few weeks).

5. Conclusion

In this paper, we have presented a new incremental minimization algorithm that replaces the traditional two-steps method - trie construction + minimization - used to build morphological lexicon by NooJ v1.x. The key of the new algorithm, which constructs the minimized FST sequentially, is a new encoding technique that links prefixed and suffixed forms to their lemma in a unified way. The new encoding boosts the minimization of the resulting FST, which leads to very compact dictionaries, even for languages that have a "heavy" morphology, such as Hungarian and Arabic.

References

- [1] J.D. Ullman and J.E. Hopcroft. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, MA, 1979.
- [2] J. Daciuk. *Incremental construction of finite-state automata and transducers, and their use in the natural language processing*. PhD thesis, Technical University of Gdańsk, Poland, 1998.
- [3] J. Graña, F.M. Barcala, and M.A. Alonso. Compilation methods of minimal acyclic finite-state automata for large dictionaries. In B.W. Watson and D. Wood, editors, *Proceedings of CIAA'2001*, pages 116–129, Pretoria, South Africa, 2001.
- [4] B.W. Watson. A taxonomy of algorithms for constructing minimal acyclic deterministic finite automata. *South African Computer Journal*, 27:12–17, 2001.
- [5] B.W. Watson. A new algorithm for construction of minimal acyclic dfas. *Science of Computer Programming*, pages 81–97, 2003.
- [6] B.W. Watson. An incremental dfa minimization algorithm. In L. Karttunen, K. Koskenniemi, and G. van Noord, editors, *Proceedings of the 2nd International Workshop on Finite State Methods in Natural Language Processing*, Helsinki, Finland, 2001.
- [7] S. Mihov and D. Maurel. Direct construction of minimal acyclic subsequential transducers. In D. Wood and S. Yu, editors, *Proceedings of CIAA'2000 conference*, pages 217–229, London, Canada, 2000.
- [8] D. Revuz. *Dictionnaires et lexiques: méthodes et algorithmes*. Ph.D. Thesis LITP 91.44, Institut Blaise Pascal, Paris, France, 1991.
- [9] S. Mihov. *Direct building of minimal automaton for given list*. PhD thesis, Bulgarian Academy of Science, 1999.
- [10] G.J. Holzmann and A. Puri. A minimized automaton representation of reachable states. *Software Tools Technol. Transfer*, 3, 1998.
- [11] B.W. Watson. A fast new semi-incremental algorithm for the construction of minimal acyclic dfas. In D. Wood and D. Maurel, editors, *Proceedings of CIAA'98 conference*, pages 91–98, Rouen, France, 1998.
- [12] D. Revuz. Dynamic acyclic minimal automaton. In D. Wood and S. Yu, editors, *Proceedings of CIAA'2000 conference*, pages 226–232, London, Canada, 2000.
- [13] R.C. Carrasco and M.L. Forcada. Incremental construction and maintenance of minimal finite-state automata. *Computational Linguistics*, 28:207–216, 2002.
- [14] B.W. Watson. A fast and simple algorithm for constructing minimal acyclic deterministic finite automata. *Universal Computer Science*, 2000.
- [15] L. Tounsi. *Sous-automates à nombre fini d'état: Application à la compression de dictionnaires électroniques*. PhD thesis, Université François Rabelais, Tours, France, 2007.
- [16] J. Daciuk. Experiments with automata compression. In D. Wood and S. Yu, editors, *Proceedings of CIAA'2000 conference*, pages 113–119, London, Canada, 2000.
- [17] J. Daciuk, S. Mihov, B.W. Watson, and R.E. Watson. Incremental construction of minimal acyclic finite state automata. *Computational Linguistics*, 26:3–16, 2000.
- [18] M. Silberztein. NooJ's dictionaries. In *Proceedings of LTC 2005*, Poznan, Poland, 2005.

Appendix A

Algorithm 2 RecCompression : Compression method used by Algorithm CompressLemma

Input: string Entry, string Lemma, string Transform

Output: string Transform

```

LongestSub  $\leftarrow$  LongestCommonSubstring(Entry, Lemma);
indEntry  $\leftarrow$  indexofLongestSubinEntry;
indLemma  $\leftarrow$  indexofLongestSubinLemma;
if indEntry  $\neq$  0 and indLemma  $\neq$  0 then
    LeftIndEntry  $\leftarrow$  LeftcontextofLongestSubinEntry;
    LeftIndLemma  $\leftarrow$  LeftcontextofLongestSubinLemma;
    Transform  $\leftarrow$  RecCompression(Transform, LeftIndEntry, LeftIndLemma);
    Transform+ = "⟨R" + (length - of - LongestSub) + "⟩";
    RightIndEntry  $\leftarrow$  RightcontextofLongestSubinEntry;
    RightIndLemma  $\leftarrow$  RightcontextofLongestSubinLemma;
    Transform  $\leftarrow$  RecCompression(Transform, RightIndEntry, RightIndLemma);
else if indEntry  $\neq$  0 and indLemma = 0 then
    Transform+ = "⟨S" + (indEntry) + "⟩";
    Transform+ = "⟨R" + (length - of - LongestSub) + "⟩";
    RightIndEntry  $\leftarrow$  RightcontextofLongestSubinEntry;
    RightIndLemma  $\leftarrow$  RightcontextofLongestSubinLemma;
    Transform  $\leftarrow$  RecCompression(Transform, RightIndEntry, RightIndLemma);
else if indEntry = 0 and indLemma  $\neq$  0 then
    Transform+ = Lemma.Substring(0, indLemma);
    Transform+ = "⟨R" + (length - of - LongestSub) + "⟩";
    RightIndEntry  $\leftarrow$  RightcontextofLongestSubinEntry;
    RightIndLemma  $\leftarrow$  RightcontextofLongestSubinLemma;
    Transform  $\leftarrow$  RecCompression(Transform, RightIndEntry, RightIndLemma);
end if
return (Transform)

```

Appendix B

Algorithm 3 LongestCommonSubstring: Longest common substring search method used by Algorithm RecCompression

Input: string Entry, string Lemma

Output: string LCS

```

n ← Entry.length;
m ← Lemma.length;
ArraySize ← Min(n, m);
firstRow = array(0..ArraySize);
secondRow = array(0..ArraySize);
maxLen ← 0;
LCS ← ""; /* LCS = Longest common substring*/
for i = 0 to n do
  for j = 2 to m do
    if Entry[i] = Lemma[j] then
      secondRow[j] ← firstRow[j − 1] + 1;
      if SecondRow[j] > maxlen then
        maxlen ← SecondRow[j];
        LCS ← "";
      else if SecondRow[j] = maxlen then
        LCS += Entry[i − maxlen + 1..i];
      end if
    else
      /* Entry[i] = Lemma[j]*/
      secondRow[j] ← 0;
    end if
  end for
  firstRow ← secondRow;
end for
return (LCS)

```

Representing and Combining Calendar Information by Using Finite-State Transducers

Jyrki NIEMI and Kimmo KOSKENNIEMI

Department of General Linguistics, University of Helsinki, Finland
e-mail: {jyrki.niemi, kimmo.koskenniemi}@helsinki.fi

Abstract. This paper elaborates a model for representing various types of semantic calendar expressions (SCEs), which correspond to the disambiguated intensional meanings of natural-language calendar phrases. The model uses finite-state transducers (FSTs) to mark denoted periods of time on a set of timelines also represented as an FST. In addition to an overview of the model, the paper presents methods to combine the periods marked on two timeline FSTs into a single timeline FST and to adjust the granularity and span of time of a timeline FST. The paper also discusses advantages and limitations of the model.

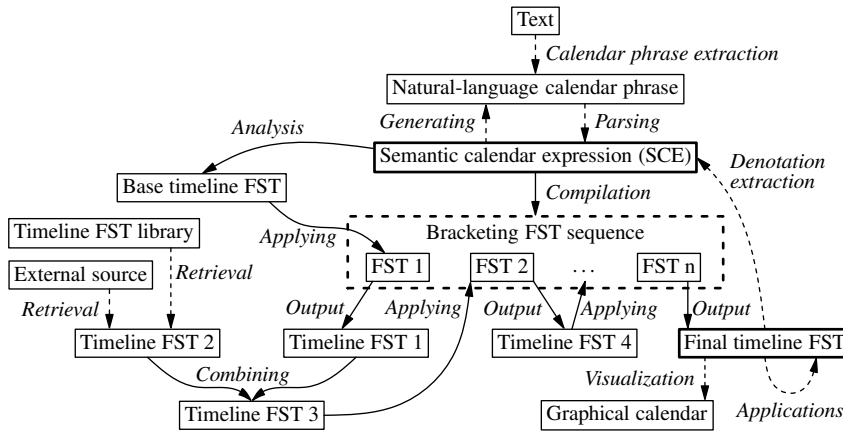
Keywords. temporal representation, calendar information, semantic calendar expressions, timeline, finite-state transducers

Introduction

Temporal information, such as calendars and schedules, is involved in many aspects of human life. The information may range from simple dates or times of day to more complex specifications; for example, a course might take place at 6–8 pm every Monday in March and April, except on Easter Monday. Such information is also used in many computer-based applications, which may present the information to a user, make computations based on it and exchange it with other applications.

Numerous models and representations have been developed for temporal information in various fields of study, from computational linguistics to temporal databases. This paper elaborates the model proposed in [1]. The model represents various types of *semantic calendar expressions* (SCEs) by using finite-state transducers (FSTs) that bracket periods of time on a set of timelines also represented as an FST. An SCE typically corresponds to the disambiguated intensional meaning of a natural-language *calendar phrase*.¹ This *bracketing FST model* can be used for temporal reasoning, in particular, for finding the common periods of time denoted by two SCEs or timeline FSTs, for example, to query an event database or to find the common free periods of several people for a meeting.

¹We use the terms *semantic calendar expression* and *calendar phrase* to correspond to *semantic term representation* and *calendar expression*, respectively, in [1].



In addition to a well-understood theoretical basis, motivations for a finite-state representation and methods include the ability to treat efficiently periodicity and certain kinds of sparse sets of sets common in calendar information.

The main contribution of this paper is a method to combine directly the periods of time denoted by two (or more) timeline FSTs into a single timeline FST, which can be further operated on (Section 2). This is useful in applications using timeline FSTs pre-constructed from SCEs, such as in finding the common free periods in several calendars, each represented by a timeline FST. The timeline FSTs to be combined should have the same granularity and span, so we also present ways to adjust these (Section 3). Furthermore, a timeline FST can be compressed to retain only the information necessary to represent the denoted periods of time (Section 4). The paper also provides an overview of the bracketing FST model (Section 1), discusses its advantages (Section 5) and limitations (Section 6), briefly reviews some related work (Section 7) and concludes with discussion and some directions for further work (Section 8).

1. The Bracketing FST Model

Figure 1 provides a conceptual overview of the various levels of representation related to the bracketing FST model and the relationships or conversions between them. The primary levels are a *semantic calendar expression* (SCE), the corresponding sequence of *bracketing FSTs* and a *timeline FST* representing the denotation of the SCE. The path from an SCE to the final timeline FST has been implemented (illustrated with solid arrows in the figure).

1.1. Semantic Calendar Expressions (SCEs)

We assume semantic calendar expressions (SCEs) as the starting point for the bracketing FST model. An SCE typically corresponds to the disambiguated intensional meaning of a natural-language calendar phrase, a possibly complex noun or prepositional phrase. However, SCEs can also represent meanings seldom expressed in natural language. The

Table 1. Examples of SCE constructs and the corresponding calendar phrases

Construct	SCE example	Calendar phrase
Calendar period	may; fri; y2008	<i>May; Friday; 2008</i>
Generic period	calday; calyear	<i>(calendar) day; (calendar) year</i>
Common part	intersect (aug, y2008)	<i>August 2008</i>
Interval	interval (may, jun)	<i>May to June</i>
List	union (mon, fri, sun)	<i>Monday, Friday and Sunday</i>
Exception	except3 (h08, mon, h09)	<i>8 am, except Mondays 9 am</i>
Anchored	nth_following (3, fri, easter)	<i>the third Friday following Easter</i>
Consecutiveness	n_consecutive (2, sun)	<i>two consecutive Sundays</i>
Parity	even_within (tue, calmonth)	<i>even Tuesdays of the month</i>
Ordinal	every_nth_within (3, fri, calyear)	<i>every third Friday of the year</i>
Containment	containing (calmonth, easter)	<i>months with Easter</i>
Quantified	n_within_each (1, mon, may)	<i>any Monday in every May</i>
Anaphoric	following (calmonth, then)	<i>the following month</i>
Deictic	containing (calday, now)	<i>today</i>

model does not reflect temporal information corresponding to verb tense and aspect or general temporal relations, for example.

An SCE may denote a specific period of time, such as 22 August 2008, or a set of periods, such as (the union of) all Mondays. SCEs of the latter kind often correspond to underspecified calendar phrases, in this case, *Monday*. The denotation of an SCE is well-defined and unambiguous on a given timeline.

Types of SCE constructs implemented in the bracketing FST model are listed in Table 1, along with the corresponding calendar phrases. Most SCE constructs can be combined with each other. For example, the SCE corresponding to *January and March 2008* (in the sense *January 2008 and March 2008*) is intersect (union (jan, mar), y2008), where jan, mar and y2008 correspond to every January, every March and the year 2008, respectively.

SCEs could be used as the primary representation of temporal information in an application. As an SCE is typically structurally close to the corresponding natural-language calendar phrase, a generation component could be used to generate the latter from the former if a natural-language presentation is needed.

If the source of temporal information is a running text, calendar phrases need to be first extracted from it and then converted to corresponding SCEs by a parser. Such conversion would be complicated by the various kinds of ambiguity and vagueness often present in natural-language calendar phrases. For example, *a week* may denote a calendar week from Monday to Sunday (or from Sunday to Saturday), any seven consecutive days or perhaps any combination of possibly disconnected periods of time whose total duration amounts to a week. The ambiguities could be avoided by presenting calendar phrases in a restricted language representing an unambiguous meaning, straightforward to convert to an SCE.

1.2. Timeline FSTs

The bracketing FST model represents time as a finite timeline string, consisting of begin and end brackets of calendar periods. The denotation of an SCE is represented by

delimiting with marker brackets the denoted periods on a timeline string. As the denotation may consist of several mutually exclusive alternatives, it is represented as an acyclic FST. Each string defined by the timeline FST represents an alternative timeline with one of the denotations marked.

A *base timeline FST* defines a single timeline string with no denotations marked, consisting of brackets and labels for calendar periods, from years down to the granularity required. The SCE to be represented is analysed to determine the granularity and span of time required for the timeline. The following is a simplified unmarked timeline string for the year 2008 at month level (spaces separate symbols):

[y y2008 [m Jan]m [m Feb]m [m Mar]m [m Apr]m ... [m Dec]m]y

To obtain the denotation of an SCE s , a base timeline FST T_b is composed with a bracketing FST (sequence) B_s representing s . B_s effectively inserts indexed marker brackets ($\varepsilon:\{in \dots \varepsilon:\}in$) into the appropriate places in the timeline string(s) defined by the timeline FST. The result $T_s = T_b \circ B_s$ is a timeline FST with the denotation of s marked in its lower language. The marker brackets are represented as transductions from the empty string ε so that a timeline FST can be used to insert the brackets to another timeline, as in combining two timeline FSTs.

A bracketing FST for a calendar period marks the calendar periods in question, whereas one corresponding to an SCE operation inserts new marker brackets based on those corresponding to the operands on the input timeline FST. The following timeline corresponds to union (jan, mar) (*January and March*):

[y y2008 $\varepsilon:\{i3 \varepsilon:\{i1 [m Jan]m \varepsilon:\}i1 \varepsilon:\}i3 [m Feb]m \varepsilon:\{i3 \varepsilon:\{i2 [m Mar]m \varepsilon:\}i2 \varepsilon:\}i3 [m Apr]m \dots [m Dec]m]y$

First, January has been marked with brackets $i1$ and March with $i2$. Then the union operation has marked with $i3$ each period marked with either $i1$ or $i2$.² The brackets $i3$ delimit the denotation of the whole expression.

To make a timeline FST smaller, the brackets inserted by the intermediate bracketing FSTs could be removed after an operation has operated on them. However, retaining at least some of the brackets could be useful in certain applications. For example, different brackets might mark different events in a calendar, and a user may want to know what events a calendar query result contains.

A timeline FST with the denotation of an SCE marked can be reused without limits in applications. It could represent complex information on service availability or one or more events with alternative realizations. To find events taking place at a certain time, the time would be marked by the corresponding bracketing FST to the timeline FST of events, followed by an intersection. Alternatively, the denotations of two timeline FSTs can be combined directly by using standard FST operations, as presented in Section 2.

It would be useful to be able to extract the denotation from a timeline FST and to represent it again as an SCE, which in turn could be generated to a natural-language calendar phrase. However, we expect finding a compact and natural SCE for a denotation marked on a timeline FST to be complicated in general.

²A sequence of adjacent brackets, such as $]m \varepsilon:\}i2 \varepsilon:\}i3$, refers to the same point of time.

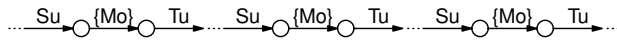


Figure 2. A timeline FST for the SCE mon denoting all Mondays; “{Mo}” denotes a marked Monday

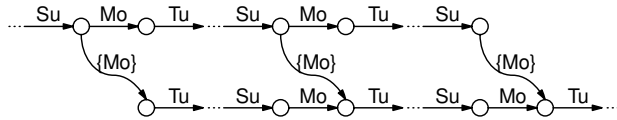


Figure 3. A timeline FST for the SCE any_n (1, mon) denoting any single Monday

1.3. Collective and Distributive Representation of SCEs

The bracketing FST model uses two types of representation for periods of time in timeline FSTs: in the *collective representation*, several different periods of time are marked on a single timeline string, whereas in the *distributive representation*, the timeline FST defines alternative timelines for different periods. We treat the representations and their relation to quantified SCEs in more detail in [2].

In the collective representation, a timeline FST defines a single timeline string, which may represent several periods of time, or equivalently, a single, possibly disconnected period of time. For example, the SCE mon denoting all Mondays is represented by having each Monday marked on a single timeline, as illustrated by the simplified timeline FST in Figure 2.³

A single timeline is insufficient for an SCE denoting possibly overlapping periods, such as *any three Mondays*, or an indefinite period which can be chosen from several alternatives, such as *any (single) Monday*. Marking all periods of three Mondays on a single timeline would in effect mark every Monday.

For such denotations, we use the distributive representation, in which a timeline FST defines a separate, alternative timeline string for each alternative denotation of the SCE. For example, the SCE any_n (1, mon) denoting any (single) Monday is represented as a timeline FST defining a set of timelines, each of which has one Monday marked, so that the timelines together cover all Mondays (Figure 3). Each alternative timeline may still contain several non-overlapping periods, as in *any three Mondays* (any_n (3, mon)).

The collective representation of periods of time is the primary one in the bracketing FST model. Accordingly, the SCE corresponding to the unquantified *Monday* is the same as that for *every Monday* (mon). In natural language, *Monday* is typically underspecified and refers to the nearest preceding or following Monday relevant in the context. A practical reason for preferring the collective representation is that it is easy for a bracketing FST to split a single timeline with every Monday marked to a set of alternative timelines, each with only one of them marked, whereas the converse operation is more complicated. The primacy of the collective representation makes the model better suited to some applications than to others, as discussed in Section 8.

³The transitions in the figures represent a number of states and transitions between them in the actual timeline FST, as the representation of each day consists of the calendar day brackets, symbols for the day of the week and day of the month, and possibly hours and even finer granularities inside.

1.4. Bracketing FSTs

An SCE is compiled to a sequence of bracketing FSTs via a sequence of compositions of regular (relation) expressions (REs). This sequence corresponds to a postfix representation of the SCE, which makes the conversion straightforward in general. The REs are represented using macros with arguments specifying the marker brackets to operate on or the label of the calendar periods to mark. After expanding the macros, the REs are compiled to the corresponding bracketing FSTs.

To obtain the marked timeline in Section 1.2, the SCE union (jan, mar) is converted to the RE macro sequence

$$\text{mon}(\text{Jan}, i1) \circ \text{mon}(\text{Mar}, i2) \circ \text{union}(i1, i2, i3).$$

The last argument of each macro indicates the brackets it inserts. The bracketing FST $\text{union}(i1, i2, i3)$ marks each January and March with brackets $i3$.⁴

Besides union for representing lists, two other basic operations in the model are intersection and interval. Intersection is used to combine periods of time of different granularities: $\text{intersect}(\text{jan}, y2008)$ corresponds to *January 2008*. The corresponding bracketing FST marks all periods that are inside both of the brackets denoting the arguments: a January inside the year 2008. Intersection is also used to find the common periods of time denoted by two SCEs. Interval represents a period of time with a specified beginning and ending period: $\text{interval}(\text{jan}, \text{mar})$ denotes all intervals from the beginning of a January to the end of the closest following March, the usual denotation of *January to March*.

2. Combining Timeline FSTs

As an alternative to applying bracketing FSTs to timeline FSTs, the denotations of two (or more) timeline FSTs of the same span and granularity can be combined directly by using common FST operations. The combined timeline FST contains the marker brackets of both the source timeline FSTs, and they can be operated on with bracketing FSTs corresponding to SCE operations, such as intersection.

When combining two timeline FSTs T_1 and T_2 , the marker brackets on them need to be represented as transductions $\varepsilon:\{in \text{ and } \varepsilon:\}in$. Let T_1 have marker brackets with index $i1$ and T_2 with $i2$. To obtain a timeline FST T_3 with both types of marker brackets in the correct places, we first insert arbitrary marker brackets $i1$ in T_2 with an insertion operation ([3], p. 5), resulting in $T_{2'}$. In effect, we allow $(\{i1 \cup \}i1)^*$ between any symbols and at the very beginning and end.⁵ $T_{2'}$ is not a well-formed timeline FST, as it may contain unpaired marker brackets. We then compose T_1 with $T_{2'}$: $T_3 = T_1 \circ (T_2 \ll (\{i1 \cup \}i1))$ (\ll denotes insertion). This effectively retains only those brackets $i1$ in $T_{2'}$ that also exist in T_1 . A combination of more than two timeline FSTs can be treated as a sequence of combinations of two timeline FSTs. The method can also be extended fairly straightforwardly to timeline FSTs containing marker brackets with several different indices.⁶

⁴Union marks each January and March on the same timeline.

⁵We would not need the multiple consecutive begin or end marker brackets allowed by the insertion operation, but they do not affect the result, either.

⁶If the two timeline FSTs contain marker brackets with identical indices, the conflicting ones must be replaced by unique ones.

We illustrate the principle with a greatly simplified timeline with symbols only for the days of the week and marker brackets. In T_2' , i^* abbreviates $(\{i1 \cup i1\})^*$. For illustration, the corresponding symbols on each timeline are aligned.

$$\begin{aligned}
 T_1 &= M \varepsilon:\{i1 \text{ Tu } W \varepsilon:\}i1 \text{ Th } F \varepsilon:\{i1 \text{ Sa } Su \varepsilon:\}i1 \\
 T_2 &= M \text{ Tu } W \text{ Th } \varepsilon:\{i2 \text{ F } Sa \varepsilon:\}i2 \text{ Su} \\
 T_2' &= i^* M \text{ } i^* \text{ Tu } i^* W \text{ } i^* \text{ Th } i^* \varepsilon:\{i2 \text{ } i^* \text{ F } i^* \text{ Sa } i^* \varepsilon:\}i2 \text{ } i^* \text{ Su } i^* \\
 T_3 &= M \varepsilon:\{i1 \text{ Tu } W \varepsilon:\}i1 \text{ Th } \varepsilon:\{i2 \text{ F } \varepsilon:\{i1 \text{ Sa } \varepsilon:\}i2 \text{ Su } \varepsilon:\}i1
 \end{aligned}$$

An intersection of $i1$ and $i2$ on T_3 would now mark the Saturday as the result.

Although the same effect can be obtained by adding marker brackets with bracketing FSTs derived from SCEs, being able to combine two timeline FSTs is an advantage in certain applications or situations. The timeline FSTs may represent possibly complex and relatively static periods of time, and constructing them might have taken a significant amount of time, so it may be more efficient to combine the timeline FSTs directly than to apply the corresponding bracketing FSTs to the other timeline FST. The timeline FSTs might also originate from different sources, and the underlying SCEs or bracketing FSTs might not be available. Moreover, a timeline FST could be based on periods of time marked in a visual calendar application, requiring an additional step to construct the corresponding bracketing FSTs or an SCE.

The ability to combine timeline FSTs also makes possible a library of predefined periods of time, for example, for holidays that are culture-specific or difficult to compute compositionally, such as Easter. Whenever an SCE refers to Easter, the timeline FST for Easter is combined with the timeline FST constructed by bracketing FSTs. The two approaches can thus be intermixed.

3. Adjusting the Span and Granularity of a Timeline FST

As the timeline FSTs to be combined must have the same span and granularity, this information has to be recorded for each timeline FST. This can be done fairly straightforwardly in the component converting SCEs to bracketing FSTs. If the spans or granularities of two timeline FSTs to be combined are different, the shorter timeline must be extended and the coarser one refined appropriately.

To combine timeline FSTs T_1 and T_2 covering a different span of time, they both are first prefixed and suffixed with any number of any symbols allowed on a timeline except marker brackets (denoted $\Sigma_{\overline{\{\}}}$): $T'_i = \Sigma_{\overline{\{\}}}^* . T_i . \Sigma_{\overline{\{\}}}^*$. The timeline FSTs T'_1 and T'_2 are then combined as described above. However, if the original timelines do not overlap, the combined timeline FST contains alternative paths for both T_1 preceding T_2 and vice versa. To retain only the correct order, we compose an unmarked reference timeline FST T_{Ref} of the appropriate span and granularity with the combined timeline FST: $T_3 = T_{Ref} \circ T'_1 \circ (T'_2 \ll (\{i1 \cup i1\}))$.

Timeline FSTs T_1 and T_2 with different granularities can be combined if we first allow any number of any symbols of the finer granularities between any symbols in the coarser-grained one.⁷ This can be achieved with the insertion operation. If T_1 is the coarser-grained timeline FST, the complete combination operation would be $T_3 = (T_1 \ll \Sigma_{\overline{\{\}}}) \circ (T_2 \ll (\{i1 \cup i1\}))$.

⁷In practice, it suffices to have $\Sigma_{\overline{\{\}}}^*$ between any symbols of the coarser timeline.

4. Compressing and Expanding a Timeline FST

The periods of time marked on a timeline FST often cover only a small part of the whole timeline. It would thus make sense to remove the parts of the timeline unnecessary for the denotation, effectively compressing the timeline. This could be useful in particular in exchanging timeline FSTs between applications. It could also make it slightly easier to verbalize the denoted periods of time.

We compress a timeline by removing each calendar period (its begin and end bracket and content) that does not contain marker brackets and that is not immediately preceded by an opening marker bracket or immediately followed by a closing one. Such unnecessary periods are removed from the coarsest to the finest granularity of the timeline.⁸ For example, the timeline in Section 1.2 representing January and March 2008 can be compressed to the following:

[y y2008 ϵ :{i3 [m Jan]m ϵ :}i3 ϵ :{i3 [m Mar]m ϵ :}i3]y

In order to be combined with other timeline FSTs or to be used as a base timeline FST on which bracketing FSTs can operate, a compressed timeline FST must be expanded again to contain all the removed calendar period brackets and symbols. To retain the denotation, the expansion of the above timeline may not insert anything between ϵ :{i3 and [m or]m and ϵ :}i3, nor any new month brackets between the existing ones. Let $\Sigma_{\{$ denote the set of opening marker brackets, $\Sigma_{\}$ the set of closing marker brackets, Σ_G the period brackets ($[_G$ and $]_G$) and other symbols for the granularity G , and t a temporary symbol. Granularity G can then be expanded with the following sequence of replace operations:⁹

$$(\epsilon \xrightarrow{1} t) \circ (t \rightarrow \epsilon \parallel \Sigma_{\{ } [_G) \circ (t \rightarrow \epsilon \parallel]_G \Sigma_{\}) \circ (t \rightarrow \epsilon \parallel [_G \cdot (\Sigma \setminus]_G)^* _) \circ (t \rightarrow \Sigma_G^*).$$

This is repeated for each granularity G from years down to the desired granularity of the timeline. A reference timeline FST is then composed with the result, which effectively inserts the marker brackets into the reference timeline FST.

5. Advantages of the Bracketing FST Model

The bracketing FST model and finite-state methods in general have a number of advantages in representing SCEs and calendar information. In general, if an SCE can be compiled to a sequence of bracketing FSTs, its denotation can be computed and represented as a timeline FST. Finite-state methods also provide a natural way to represent cycles and repetition typical of calendar information.

A timeline FST provides a compact representation for certain kinds of sets of sets. For example, on a timeline of one year, *(any) five days a year* corresponds to $\binom{365}{5}$, or more than 5×10^{10} , different alternative timelines, each with a different combination of five days, but the corresponding timeline FST is only about eleven times as large as that

⁸Weeks cannot be removed similarly, since removing a week would also remove the beginning or ending bracket of a month or a year occurring within the week.

⁹The first replace operation $\epsilon \xrightarrow{1} t$ is restricted so that it inserts only a single t between each two symbols in the source string.

for a single day. Each state in this timeline FST encodes the number of marked days that far but not the exact days.

A timeline FST can encode a set of alternative timelines. In contrast, a logic programming approach would typically return one alternative at a time and backtrack to find the next one. In some applications that may be preferable, in particular if the result set would be very large, whereas in others we might want to enumerate all the alternatives.

Once constructed, an FST can be reused. For example, an application may make different queries to a timeline FST or directly combine two timeline FSTs and operate on them.¹⁰ In contrast, a typical logic programming approach might interpret the source expression anew for each new query.

6. Limitations of the Bracketing FST Model

FSTs are a simple model of computation, and the bracketing FST model shares the limitations of finite-state methods, including the inefficiency of some constructions and the limited calculation ability.

Although the finite-state representation is often relatively efficient, it has its limitations. Even if the final timeline FST corresponding to an SCE were representable, constructing it might consume too much space and time to be practically feasible. In the worst case, the size of the result of a composition of FSTs may be the product of the sizes of the composed FSTs, and even if the size of the result were reasonable, computing the composition may be intractable. An example is the bracketing FST sequence representing the meaning *a period of two days, a period of three days and a period of four days, not overlapping*.

FSTs can count and perform arithmetic operations only if limited to a finite and relatively small set of numbers. The complexity of the required FSTs increases with the set of numbers, as each case has to be enumerated separately. In practice, the counting ability is sufficient for typical calendar information. Counting the number of marked periods or computing their duration beyond a fixed limit would require a simple external facility.

SCEs referring to a fixed number of periods, such as $n_consecutive$ (10, calday) (*ten consecutive days*), are represented by concatenating the same basic RE (or bracketing FST) the specified number of times. The power notation R^n of many RE formalisms is only an abbreviation for n REs R concatenated.

Because of the inability to count in a general manner, meanings referring to equivalences of different kinds must be represented by enumerating each possibility. For example, the meaning *the same day of the week every week* would have to be represented as *every Monday or every Tuesday or ... or every Sunday*. The same holds for meanings involving equality of length, such as *5–10 days a month, each month the same number of days*.

Some features of calendars involve complex computations. For example, although it might be possible to calculate the date of Easter with finite-state methods, it would be very awkward and tedious. However, in the applications we envisage for the model, it would suffice to use precomputed dates for Easter.

¹⁰Although making queries using intersection effectively constructs a modified timeline FST, it need not be completely reconstructed.

7. Related Work

Temporal representation and reasoning have been widely studied in various fields for different purposes. Finite-state representations of temporal information include regular expressions for checking the validity of dates by Karttunen et al. [4] and the representation of events combined with temporal information by Fernando [5]. However, they are rather different in purpose from the bracketing FST model. We now briefly describe some research that is related to the bracketing FST model in purpose or coverage, though not in representation.

Endriss [6] proposes Temporal Expression Language (TEL) for representing and reasoning with temporal expressions in appointment negotiation dialogues in the Verbmobil project [7]. A TEL expression ultimately denotes a single, connected period of time, but underspecification is represented as a set of possible periods. A prototype of TEL was implemented in Prolog. Endriss notes that a major reason for the inefficiency of the prototype was representing (underspecified) periods of time as lists of intervals and intersecting such lists ([6], p. 150).

Han and Lavie [8] propose Time Calculus for Natural Language (TCNL) for representing and reasoning with temporal expressions and chains of them in text. TCNL treats temporal focus and underspecification. Reasoning is implemented as temporal constraint satisfaction; it can be used in such applications as anchoring temporal expressions in text on the basis of a chain of expressions referring to each other. Although solving general temporal constraint satisfaction problems is NP-complete, many types of problems can be solved in polynomial time.

The Calendar Logic of Ohlbach and Gabbay [9] is a propositional temporal logic in which operations can refer to points or periods of time represented symbolically. It can encode the meaning of various types of natural-language expressions involving temporal information and events. The satisfiability of a Calendar Logic formula can be decided by translating it to propositional logic, which may increase its size exponentially, or by using a tableau system.

The CTTN system (Computational Treatment of Temporal Notions) of Ohlbach [10] is used to model periodic temporal notions, from timetables and conference dates to whole calendar systems. CTTN represents different kinds of time points and intervals, including fuzzy ones, along with partitionings of a timeline. Operations on these notions include fuzzy set operations and interval relations. CTTN has its own functional specification language GeTS for defining temporal notions. CTTN can model real-world temporal phenomena from leap seconds and time zones to the historical succession of calendar systems.

TimeML [11] is a markup language for marking events and temporal expressions and relations in text. In contrast to SCEs, TimeML often represents the explicit denotation of a temporal expression instead of its intensional meaning. Dale and Mazur [12] propose within the TimeML framework a representation that retains the underspecification of such expressions as *Monday*. Their representation is based on attribute-value matrices but they also provide a compact encoding as an extension of the ISO date and time format.

OWL-Time [13] is an ontology for representing temporal information on the Web in particular. The ontology can be represented in first-order predicate logic. It can represent the meaning of various kinds of temporal expressions, including repeated events (disconnected periods) using temporal aggregates [14].

8. Discussion and Further Work

We think that directly combining the calendar information marked on two timeline FSTs would be useful in such applications as finding common free periods in the calendars of several people. The ability to reuse and operate on preconstructed timeline FSTs would appear an advantage of the bracketing FST model. Calendar information could be exchanged between applications as compressed timelines.

Although the bracketing FST model can represent a number of different calendar information constructs, some fairly common ones remain untreated. We think that it would be important to be able to represent at least a useful subset of durations. In general, a duration, such as *40 hours*, might be composed of an arbitrary number of arbitrarily short periods of time, so in practice, it cannot be represented by marking all possible alternatives on a set of timelines. Also untreated are shifting constructs, such as *five months after 10–16 June*, interpreted as the period of 10–16 June shifted five months forward.

Calendar information in general can be very complex and impossible to represent exactly or naturally with finite-state methods. However, we should try to find a reasonable set of operations for representing the meaning of most calendar phrases in relevant application domains. Some constructs might be treated in the conversion process from an SCE to a bracketing FST sequence.

The collective representation is well suited for representing a single set of periods of time (or equivalently, a single disconnected period) denoting the times provided for a service or available for an activity, from which a user can choose one or more. Instead of a single solution, a timeline FST represents a set of periods suitable for a purpose, whereas a typical logic programming approach might also in this case enumerate the available periods, though one at a time. Applications that could use the collective representation include querying an event calendar to find an event at a specified time, making an appointment to a service, given certain free periods of time, and scheduling a time for a meeting among a number of people with their respective calendars.

In contrast, the distributive representation is required when a user would like to find a period or set of periods among many that fulfill certain criteria, such as the possible working hours during a week. The distributive representation effectively represents all the different sets of periods of time fulfilling the criteria, instead of only one, as typically requested. Even though a timeline FST can represent certain types of large sets of periods relatively compactly, constructing such timeline FSTs by a composition of bracketing FSTs may be too inefficient. In such applications, a Prolog-style approach returning only one solution, and more only by request, would in general be more efficient.

In summary, we think that the bracketing FST model is a promising representation for the semantics of many types of calendar phrases represented as SCEs. Being able to find the common periods of time denoted by two SCEs or the corresponding timeline FSTs could be useful in various applications. However, to be usable in practice, the model needs further work both on widening its coverage and on improving its performance. Moreover, practical applications would benefit from a component to parse a (restricted) natural-language calendar phrase to an SCE and another one to generate the former from the latter.

Acknowledgements

This paper represents independent work by the first author based on the suggestions of the second author. The work was funded by the Graduate School of Language Technology in Finland. We thank the anonymous reviewers for their valuable comments. We are also grateful to Anssi Yli-Jyrä for pointing out how to convert a distributive representation to a collective one.

References

- [1] Jyrki Niemi and Kimmo Koskenniemi. Representing calendar expressions with finite-state transducers that bracket periods of time on a hierarchical timeline. In Joakim Nivre, Heiki-Jaan Kaalep, Kadri Muischnek, and Mare Koit, editors, *Proceedings of the 16th Nordic Conference of Computational Linguistics NODALIDA-2007*, pages 355–362, Tartu, Estonia, 2007. University of Tartu.
- [2] Jyrki Niemi and Kimmo Koskenniemi. Quantification and implication in semantic calendar expressions represented with finite-state transducers. In *22nd International Conference on Computational Linguistics (Coling 2008): Companion volume: Posters and Demonstrations*, pages 69–72, Manchester, UK, August 2008. Coling 2008 Organizing Committee.
- [3] Helmut Schmid. SFST manual. <ftp://ftp.ims.uni-stuttgart.de/pub/corpora/SFST/SFST-Manual.pdf>, September 2007.
- [4] L[auri] Karttunen, J[ean]-P[ierre] Chanod, G[regory] Grefenstette, and A[nne] Schiller. Regular expressions for language engineering. *Natural Language Engineering*, 2(4):305–328, December 1996.
- [5] Tim Fernando. A finite-state approach to events in natural language semantics. *Journal of Logic and Computation*, 14(1):79–92, 2004.
- [6] Ulrich Endriss. Zeitliche Ausdrücke in Terminvereinbarungsdialogen: Repräsentation und Inferenz. Diplomarbeit, Technische Universität Berlin, Fachbereich Informatik, June 1998.
- [7] Wolfgang Wahlster, editor. *VerbMobil: Foundations of Speech-to-Speech Translation*. Artificial Intelligence. Springer, Berlin, 2000.
- [8] Benjamin Han and Alon Lavie. A framework for resolution of time in natural language. *ACM Transactions on Asian Language Information Processing (TALIP)*, 3(1):11–32, March 2004.
- [9] Hans Jürgen Ohlbach and Dov Gabbay. Calendar logic. *Journal of Applied Non-classical Logics*, 8(4):291–324, 1998.
- [10] Hans Jürgen Ohlbach. Computational treatment of temporal notions: The CTTN-system. In Frank Schilder, Graham Katz, and James Pustejovsky, editors, *Annotating, Extracting and Reasoning about Time and Events*, volume 4795 of *Lecture Notes in Computer Science*, pages 72–87. Springer, 2007.
- [11] Roser Saurí, Jessica Littman, Bob Knippen, Robert Gaizauskas, Andrea Setzer, and James Pustejovsky. TimeML annotation guidelines, version 1.2.1. http://timeml.org/site/publications/timeMLdocs/annguide_1.2.1.pdf, January 2006.
- [12] Robert Dale and Paweł Mazur. The semantic representation of temporal expressions in text. In Mehmet A. Orgun and John Thornton, editors, *Australian Conference on Artificial Intelligence*, volume 4830 of *Lecture Notes in Computer Science*, pages 435–444. Springer, 2007.
- [13] Jerry R. Hobbs and Feng Pan. An ontology of time for the semantic web. *ACM Transactions on Asian Language Information Processing (TALIP)*, 3(1):66–85, March 2004.
- [14] Feng Pan and Jerry R. Hobbs. Temporal aggregates in OWL-Time. In *Proceedings of the 18th International Florida Artificial Intelligence Research Society Conference (FLAIRS-2005)*, pages 560–565, Clearwater Beach, Florida, 2005. AAAI Press.

Optimality Theory and Vector Semirings¹

Wolfgang SEEKER, Daniel QUERNHEIM

Institut für Linguistik, Universität Potsdam

Karl-Liebknecht-Str. 24–25, D-14476 Golm

seeker@ling.uni-potsdam.de

Daniel.Quernheim@uni-potsdam.de

Abstract. As [1] and [2] have shown, some applications of Optimality Theory can be modelled using finite state algebra provided that the constraints are regular. However, their approaches suffered from an upper bound on the number of constraint violations. We present a method to construct finite state transducers which can handle an arbitrary number of constraint violations using a variant of the tropical semiring as its weighting structure. In general, any Optimality Theory system whose constraints can be represented by regular relations, can be modelled this way. Unlike [3], who used roughly the same idea, we can show, that this can be achieved by using only the standard (weighted) automaton algebra.

Keywords. Semiring, Optimality Theory, Weighted Finite State Algebra

Introduction

Optimality Theory (OT) has been a popular framework since it was first developed in the early 1990s by [4], modelling phenomena in phonology, syntax and other linguistic disciplines. The computational complexity of OT itself has been shown to be within the finite state domain ([1]), while the complexity of a particular OT system depends on the constraints that are assumed. One problem in earlier analyses was how to handle multiple constraint violations ([2]). This has been addressed by using weighted finite automata ([5,6,3]), though these proposals were not fully formalized and introduced specially-designed algorithms instead of making use of well-known properties of finite state algebra.

In this paper, we present a formal account of OT and the closely-related Harmonic Grammar and introduce what we call the *tropical vector semiring* which allows us to model OT with plain finite state algebra without the need for *ad hoc* algorithms. Subsequently, we illustrate our proposal with the implementation of an OT analysis of assimilation phenomena in Dutch.

¹For our implementation, we used FSM<2.0> by Thomas Hanneforth, which can be downloaded from <http://www.ling.uni-potsdam.de/~tom/fsm/>. We would like to thank four anonymous reviewers for valuable feedback.

1. Preliminaries

In this section we will briefly review the notions of semirings and weighted finite state automata and give a formal definition of Harmonic Grammar and Optimality Theory.

1.1. Semirings and Weighted Automata

Weighted automata differ from their unweighted counterparts in that their transitions contain weights in addition to the usual alphabet symbols. In order to allow for the various operations that are defined for weighted automata, the used weight set must have the algebraic structure of a semiring. As is shown in [7], cleverly designed weighted automata can even be used to recognize clearly context free languages. Weighted automata are indeed a more general device than unweighted automata, since every unweighted automaton can be represented by a weighted automaton that uses the so called boolean semiring as its weighting structure. A semiring is defined in the following way (cf. [8]):

Definition 1. A right semiring is a 5-tuple $\langle \mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1} \rangle$, where

1. $\langle \mathbb{K}, \oplus, \bar{0} \rangle$ is a commutative monoid with $\bar{0}$ as its identity element,
2. $\langle \mathbb{K}, \otimes, \bar{1} \rangle$ is a monoid with $\bar{1}$ as its identity element,
3. \otimes right distributes over \oplus : $\forall a, b, c \in \mathbb{K}, (a \otimes b) \oplus c = (a \otimes c) \oplus (b \otimes c)$,
4. $\bar{0}$ is an annihilator for \otimes : $\forall a \in \mathbb{K}, a \otimes \bar{0} = \bar{0} \otimes a = \bar{0}$.

A left semiring is defined likewise by replacing right with left distributivity. A semiring, that fulfills both conditions is simply called semiring. If the second monoid is commutative as well, the semiring is called a commutative semiring. This is an important feature in real applications, since intersection and hence composition is defined for weighted automata only if their weighting structure is a commutative semiring². An often used semiring is the so called tropical semiring $\langle \mathbb{R}, \min, +, \infty, 0 \rangle$, that computes the minimal weight for a given input.

A weighted finite automaton over the semiring \mathbb{K} is defined as the 7-tuple $A = \langle \Sigma, Q, I, F, E, \lambda, \rho \rangle$, with Σ as its alphabet, Q as the set of states, $I \subseteq Q$ as the set of start states, $F \subseteq Q$ as the set of final states, $E \subseteq Q \times \Sigma \times \mathbb{K} \times Q$ as the set of transitions, $\lambda : I \rightarrow \mathbb{K}$ as the initial weight function and $\rho : F \rightarrow \mathbb{K}$ as the final weight function. A path $\pi = e_1 \dots e_n$ in A is an element of E^* with consecutive transitions. The weight of a path $w[\pi] = w[e_1] \otimes \dots \otimes w[e_n]$ is then the \otimes -product of the weights of its constituent transitions. The output weight of a given string x applied to the automaton A is defined as follows, where $\Pi(x)$ is the set of paths, that are labeled with x , $s[\pi] \in I$ is the source state of π and $t[\pi] \in F$ is the target state of π :

Definition 2. $A \cdot x = \bigoplus_{\pi \in \Pi(x)} \lambda(s[\pi]) \otimes w[\pi] \otimes \rho(t[\pi])$

If the set of paths for x is empty, its weight is defined as $\bar{0}$. Paths with weight $\bar{0}$ are usually seen as non existent in the automaton³. This definition can easily be extended to weighted ϵ -automata and transducers.

²This restriction holds only for non trivially weighted automata, that is, automata which contain weights different from $\bar{1}$.

³This is a problem for the proposed constructions in [7], if one tries to map certain paths of an automaton to a weight, that is simultaneously the $\bar{0}$ of the weighting semiring.

1.2. Harmonic Grammar and Optimality Theory

The framework of Optimality Theory (OT) was introduced by [4] as a further development of Harmonic Grammar (HG, [9]), which resulted from research in neural networks. OT is widely used in phonology, syntax and other subdisciplines of linguistics. The basic ideas behind OT and HG are the notions of the ‘*richness of the base*’ and ‘*constraint violability*’. This means that there is a generating function which maps an input to an unrestricted number of candidates (which do not necessarily have anything in common with the input). These candidates are then evaluated using a set of constraints which are either ranked (OT) or weighted (HG). Ideally, constraints are the same for all languages. Cross-linguistic variation is then explained by different rankings/weightings.

While in OT every constraint only leaves the candidates with the least number of violations, multiple violations of lower-weighted constraints can outrank higher-weighted constraints in HG. If, for instance, you have two constraints weighted 2 and 1, a candidate that violates the latter three times (score: 3) will be inferior to one that violates only the former once (score: 2). In an OT system with separately ranked constraints, this is not possible.

There has been a lot of research on the computational properties of OT. It has been shown by [1], [2] and others that an OT system can be modelled using finite state algebra provided the constraints can be expressed by regular operations. In order to model unlimited constraint violations, one has to equip these automata with a weighting structure ([6,5,3]). Most of the work so far has focused on this, leaving aside the issue of how the actual constraint automata look. In the following, we will provide some examples as well as general implementation instructions, using only finite state algebra.

Let us now come to a more formal description of OT and HG. OT has been well formalized by [10], but we will take a slightly different approach here, formalizing HG first and developing a formalization of OT based on this. Both theories are closely related in that they have the same basic structure, consisting of a generator function GEN and an evaluating function EVAL which makes use of a finite constraint sequence $C = (C_1, C_2, \dots, C_n)$.

GEN is a function mapping the input on an unrestricted number of candidates. In order for it to be as general as possible, we have it operate on the closure of two given alphabets Σ and Θ (e.g. the set of phonemes and the set of phones):

$$\text{GEN} : \Sigma^* \rightarrow 2^{\Theta^*}$$

Every constraint C_n is a function mapping the input⁴ and a candidate to a natural number signifying the number of violations:

$$C_n : \Sigma^* \times \Theta^* \rightarrow \mathbb{N}$$

To keep track of the weights in HG, we further define the function $w : \mathbb{N} \rightarrow \mathbb{R}$ which maps every n to the weight of the corresponding constraint C_n . For OT, this is trivially 1 for every constraint.

⁴Whether a constraint actually refers to the input or not marks the difference between *markedness constraints* (which penalize dispreferred structures) and *faithfulness constraints* (which demand for some correspondence between input and output). We will discuss how to unify them later.

The evaluating function is what differentiates OT from HG. We define the function $\text{EVAL}_{\text{HG}} : \Sigma^* \times \Theta^* \rightarrow \mathbb{R}$ that marks candidates in the following way:

$$\text{EVAL}_{\text{HG}}(\text{in}, z) = \sum_{i=1}^n C_i(\text{in}, z) \cdot w(i)$$

In order to determine the optimal candidate, the one with the lowest value of EVAL_{HG} has to be found, that is:

$$\text{BEST}(\text{in}) = \underset{z \in \text{Gen}(\text{in})}{\text{argmin}} \text{EVAL}_{\text{HG}}(\text{in}, z) = \underset{z \in \text{Gen}(\text{in})}{\text{argmin}} \sum_{i=1}^n C_i(\text{in}, z) \cdot w(i)$$

It is obvious that the latter formula is an instantiation of definition 2 where the abstract multiplication \otimes is the sum and the abstract addition \oplus is the min operation, with λ and ρ being the identity element of \otimes . In fact, this weighting structure is the tropical semiring.

It is easy to see that an OT system with only one allowed violation per constraint can be imitated by a HG that weights constraints sufficiently far apart, e.g. in powers of 2. However, when multiple violations are allowed, this is not possible anymore, because for every pair of weights $\langle a, b \rangle$ for the constraints A, B there would be a number of violations c which can make B outrank A . For OT, a vector of constraint violations has to be constructed by EVAL_{OT} , so that the values of individual constraint violations cannot interfere with each other:

$$\text{EVAL}_{\text{OT}}(\text{in}, z) = \langle C_1(\text{in}, z), C_2(\text{in}, z), \dots, C_n(\text{in}, z) \rangle.$$

We want to mention that OT can be seen as a vector of trivial Harmonic Grammars of one constraint each, each yielding a result for every candidate. Given that OT can have equally ranked constraints, this would mean Harmonic Grammars of more constraints that are weighted the same. This way, one could even create grammars of independent systems of differently ranked constraints, an idea that (to our knowledge) hasn't been explored yet.

In order to have an ordering on violation markings, the vector result of EVAL_{OT} has to be interpreted lexicographically; that means just as in OT a candidate loses to another one if it has more violations for the first constraint they differ in, the vectors are compared item by item. The next section will give the formalization of the underlying algebra.

2. The Tropical Vector Semiring

As mentioned before, a strict ordering of constraints cannot be achieved by a weighting structure that simply computes the sum of all weights (i.e. constraint violations). Therefore we define the support set of the weighting structure to be the set of vectors, where every dimension of the vector keeps track of a certain constraint. Adding two weights then means to sum the individual values in the same dimensions, which is of course simple vector addition. Thus, an output candidate of the OT system is weighted by a vector that represents in each dimension the number of violations for the corresponding constraint.

Definition 3. The tropical vector semiring $\langle \mathbb{N}_0^d \cup \{\vec{\infty}\}, \min, +, \vec{\infty}, \vec{0} \rangle$ is a 5-tuple, where

1. its support set \mathbb{N}_0^d is the set of vectors with a fixed dimension d ,
2. \oplus is the minimum operation returning the smaller one of two given vectors.
3. \otimes is vector addition, which is defined as pairwise addition of its components:

$$\vec{a} + \vec{b} = \begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix} + \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix} = \begin{pmatrix} a_1 + b_1 \\ \vdots \\ a_n + b_n \end{pmatrix},$$

4. $\vec{0}$ is the infinite vector $\vec{\infty}$, that has in every dimension an infinite value,
5. $\vec{1}$ is the null vector $\vec{0}$.

The minimum operation is supposed to determine the smaller one of two given vectors, where the meaning of smaller differs from the usual meaning of being shorter than the other vector. We impose a strict partial order on the support set (i.e. the set of vectors) by ranking the dimensions of a vector:

$$\vec{a} < \vec{b} \stackrel{\text{def}}{=} \begin{cases} \text{false} & \text{if } \vec{a} = \vec{b} \\ \text{true} & \text{if } a_1 < b_1 \\ \text{false} & \text{if } b_1 < a_1 \\ \begin{pmatrix} a_2 \\ \vdots \\ a_n \end{pmatrix} < \begin{pmatrix} b_2 \\ \vdots \\ b_n \end{pmatrix} & \text{if } a_1 = b_1 \end{cases}$$

The minimum operation is defined as follows:

$$\vec{a} \min \vec{b} \stackrel{\text{def}}{=} \begin{cases} \vec{a} & \text{if } \vec{a} < \vec{b} \\ \vec{b} & \text{if } \vec{b} < \vec{a} \\ \vec{a} & \text{otherwise} \end{cases}$$

Thus, a vector will be smaller than another one, if it has a smaller value in the first dimension they differ in. In the case of two completely equal vectors, we arbitrarily define the operation to return the first one. With this feature we can represent an unrestricted number of constraint violations for a single constraint and obey the constraint ranking at the same time.

It is easy to show that $\langle \mathbb{N}_0^d \cup \{\vec{\infty}\}, \min, \vec{\infty} \rangle$ and $\langle \mathbb{N}_0^d \cup \{\vec{\infty}\}, +, \vec{0} \rangle$ are both commutative monoids. The \oplus operation is by definition idempotent. Moreover, the following monotony property for the \otimes operation (vector addition) holds:

$$\forall \vec{a}, \vec{b} : \forall \vec{c} \neq \vec{\infty} : \vec{a} < \vec{b} \text{ implies } \vec{a} + \vec{c} < \vec{b} + \vec{c}$$

This is due to the fact that component-wise addition of natural numbers preserves monotony for every component.⁵ This allows for the following proof of right distributiv-

⁵One reviewer has pointed out that distributivity would be lost when allowing for ∞ as a component: $\left(\begin{pmatrix} 1 \\ 2 \end{pmatrix} \oplus \begin{pmatrix} 2 \\ 1 \end{pmatrix}\right) \otimes \begin{pmatrix} \infty \\ 1 \end{pmatrix} = \begin{pmatrix} \infty \\ 3 \end{pmatrix}$, but $\left(\begin{pmatrix} 1 \\ 2 \end{pmatrix} \otimes \begin{pmatrix} \infty \\ 1 \end{pmatrix}\right) \oplus \left(\begin{pmatrix} 2 \\ 1 \end{pmatrix} \otimes \begin{pmatrix} \infty \\ 1 \end{pmatrix}\right) = \begin{pmatrix} \infty \\ 2 \end{pmatrix}$. That is why we restrict components to natural numbers (with the obvious exception of the infinite vector). Thus, the number of constraint violations in OT may be unbounded, but not infinite.

ity (left distributivity can be proven analogously):

$$(\vec{a} \oplus \vec{b}) \otimes \vec{c} = (\vec{a} \otimes \vec{c}) \oplus (\vec{b} \otimes \vec{c})$$

Proof. If $\vec{c} = \vec{\infty}$, the result is $\vec{\infty}$. If $\vec{a} = \vec{b}$, the result is $\vec{a} \otimes \vec{c}$. If they are unequal, without loss of generality (since the min operation is commutative) assume that $\vec{a} < \vec{b}$. Then $\vec{a} \oplus \vec{b} = \vec{a} \min \vec{b} = \vec{a}$, which gives:

$$(\vec{a} \oplus \vec{b}) \otimes \vec{c} = \vec{a} \otimes \vec{c}$$

Due to monotony, we know that $\vec{a} + \vec{c} < \vec{b} + \vec{c}$, which yields:

$$(\vec{a} \otimes \vec{c}) \oplus (\vec{b} \otimes \vec{c}) = \vec{a} \otimes \vec{c}$$

qed.

The tropical vector semiring is therefore commutative, idempotent and bounded and allows for the intersection and composition of weighted finite state acceptors and transducers.

[11] has achieved a similar result by composing several semirings and then redefine the \oplus -operation in order to work globally on the composite semiring. In his formalisation, the support set of the resulting semiring is interpreted as a set of tuples with arity of the number of composed semirings.

3. The Model

One advantage of modelling OT with finite state machines is the possibility to compile the whole system into one monolithical transducer, as has been already pointed out in [3]. Since every part of the system is represented by a finite state transducer (or some identity transduction of an underlying automaton), the composition (\circ) of the individual parts will give us a single transducer for the whole system (cf. [12]).

Definition 4. OT-transducer (OTt)

$$\begin{array}{c} \text{GEN} \\ \circ \\ \text{EVAL} \\ \circ \\ \text{Postprocessing} \end{array}$$

The problem of finding the optimal candidate for a given input is therefore the problem of finding the best path in the OTt for the input. In the tropical vector semiring, the best path is defined as the path with the smallest vector, with the vector representing the violations of this candidate. This problem can also be seen as a search problem in vector space and this could be another application for the tropical vector semiring.

Definition 5. *Finding the optimal candidate*

$$\text{BEST}(\text{input}) = \text{best_path}(\text{ID}(\text{input}) \circ \text{OTt})$$

One difficulty of modelling OT with finite state tools is the definition of a special class of faithfulness constraints, namely those which require some kind of identity between input and output. Since it is not possible to link two positions on different tapes of a transducer (alignment problem), the identity of input and output is hard to test. One possible way of addressing this is to use a special symbol different from epsilon to mark deletion or epenthesis, so that optimization algorithms, that for example remove or push epsilons, cannot destroy the correspondence. These transducers meet the same length condition (see [12]).

However, we want to propose another solution to this problem. In order to establish a correspondence between input and output segments, we lift the content of the lower tape onto the upper tape and mark it with a special symbol (§). An input-output pair from the language of the OTt is then no longer represented by a two taped transducer, but is now an automaton, that has some sort of complex symbols on its tape⁶. The advantage of this approach is the direct applicability of our constraints, which we model by regular relations, that reweight § in the context of constraint violations. With this notation, every (regular) faithfulness constraint can be modelled in the same way as markedness constraints. Example 1 shows the Dutch word *eetbar* /etbar/ ⁷ that was mapped by GEN on the candidate *edbar*. The . marks syllable boundaries and the curly brackets enclose prosodic words.

Example 1.

$$\{ (e\$e) (d\$t) \} . \{ (b\$b) (a\$a) (r\$r) \}$$

To model epenthesis and deletion as well, GEN has to introduce segments of the form ([Phone] §) for deletion and (§ [Phone]) for epenthesis, where the output or input symbol is the empty word. A constraint like MAX-IO(no deletion) or DEP-IO(no epenthesis, see e.g. [13]) would then punish sequences of §) or (§.

4. Implementation

4.1. Data: Dutch Assimilation

The OT analysis we will use as an example models assimilation in Dutch and was developed by [14]. Additional test data was drawn from [13].

Dutch exhibits final devoicing as in *baard* [ba:rt] ‘beard’. Additionally, obstruents assimilate when words are derived (e.g. by compounding, examples from [13]): *stropdas* [-bd-] ‘tie’, *diepzee* [-ps-] ‘deep-sea’. As can be seen, there is progressive assimilation as well as regressive assimilation, for which the following patterns can be observed:

⁶This idea is also used in transducer optimization algorithms, that encode the symbol pairs of a transition in one complex symbol forming thus an automaton. These automata can then be minimized and afterwards decoded in order to get back the improved transducer.

⁷This is supposed to be an underlying representation; vowel length is omitted for the sake of simplification.

Example 2. Dutch assimilation data ([13])

1. *voiceless stop + voiced stop: regressive* (zakdoek [-gd-] ‘handkerchief’)
2. *voiced stop + voiceless stop: regressive* (bloedkoraal [-tk-] ‘red coral’)
3. *voiceless fricative + voiced stop: regressive* (lafbek [-vb-] ‘coward’)
4. *voiced fricative + voiceless stop: regressive* (hooxtij [-yt- / -kt-] ‘heyday’)
5. *voiceless stop + voiced fricative: progressive* (haartzeer [-ts-] ‘heartache’)
6. *voiceless fricative + voiced fricative: progressive* (straafzaak [-fs-] ‘trial’)
7. *voiced stop + voiced fricative: both become voiceless* (handvat [-tf-] ‘handle’)
8. *voiced fricative + voiced fricative: both become voiceless* (hoogvlakte [-yf- / -xf-] ‘plateau’)

Some generalizations can be drawn from this data: First, when words are compounded, the segments at the boundary always share the same voice feature. Second, stops in the onset always keep their voice feature and always assimilate their neighbouring segment. Third, there seems to be a general dispreference of voice at the end of the prosodic word which is in correspondence with general devoicing in Dutch. From this we can deduce three constraints (due to [14]) which give an account of the data:


Definition 6. S-IDENT: *Adjacent obstruents are identical in voicing.*

Definition 7. IDENT-PWOS: *Stops in onset position of Prosodic Words should be faithful to underlying laryngeal specification.*

Definition 8. $\{+[VOICE]\}_\omega$: *Prosodic Word-final obstruents are voiceless.*

From example 2.1 we know that S-IDENT dominates $\{+[VOICE]\}_\omega$, because otherwise, we’d expect [-kd-]. How is IDENT-PWOS ranked with respect to them? If it was dominated by $\{+[VOICE]\}_\omega$, in example 2.3 a candidate with [-fp-] would be better *ceteris paribus*. No ranking between S-IDENT and IDENT-PWOS can be deduced; following [14] we will therefore assume them to be at the same level. A tableau for the input *etbar* /etbar/ [-db-] ‘edible’ is given in table 1.

Table 1. OT tableau for /{et}.{bar}/

/ {et} . {bar} /	S-IDENT	IDENT-PWOS	$\{+[VOICE]\}_\omega$
a.  [ed.bar]			*
b. [et.bar]	*!		
c. [ed.par]	*!	*	*
d. [et.par]		*!	

Some remarks have to be made about this tableau. First, we assume that the input is already supplied with a structure that at least indicates prosodic words and syllables. In practice, the generator function would introduce arbitrary structures from which other constraints would pick the adequate ones; but this would only complicate our analysis unnecessarily. Second, we use rougher approximations of the phonetic realisations than [14] do, since this is sufficient for present purposes. Third, the constraints so far are not sufficient to exclude unwanted candidates like [eg.bar] and even [eg.ber] which would incur the exact same constraint violations as the winning candidate [ed.bar]. We therefore add the following constraints to our analysis (from [13]):

Definition 9. IDENT-IO(PLACE): *The specification for place of articulation of an input segment must be preserved in its output correspondent.*

Definition 10. IDENT-IO: *Corresponding segments are exactly the same in input and output.*

We rank IDENT-IO(PLACE) highest and IDENT-IO lowest to obtain the tableau shown in table 2. Our example does not use epenthesis or deletion for it would complicate the constraint set even more.

Table 2. OT tableau for /{et}.{bar}/

{et}.{bar}/		IDENT-IO(PLACE)	S-IDENT	IDENT-PWOS	*[+VOICE] _ω	IDENT-IO
a.	ed.bar				*	*
b.	eg.bar	*!			*	*

4.2. The GEN Function as Regular Relation

Since the GEN function is supposed to generate all possible candidates for a given input, we want to define it as unrestricted as possible. Finite state automata are capable of representing infinite languages and are therefore a natural way to represent the possibly infinite set of candidates generated by GEN. In many cases, however, GEN can be restricted in a way that leads to a finite, but still complete set of reasonable candidates. In our Dutch example, GEN has to generate all possible substitutions of consonants, that can appear at syllable boundaries. But still, to avoid an oversimplified GEN function, we define it as the set of pairs of phonemes, i.e. the crossproduct of the set of phonemes with itself⁸.

Definition 11. GEN

$$[\text{Phone}] @ \rightarrow \backslash (\dots \$ [\text{Phone}] \backslash)$$

This regular expression compiles to a transducer that uses longest match to enclose every member of the set of phones with a (on the left side and \$[Phone]) on the right side (see [15]). The double occurrence of [Phone] thereby models the crossproduct $[\text{Phone}] \times [\text{Phone}]$. This definition of GEN is thus very general, but produces a lot of unnecessary and unreasonable candidates. The two constraints IDENT-IO(PLACE) and IDENT-IO will help us with that. In addition to the generation of the candidate set, the GEN function introduces the previously described notation, that enables us to model faithfulness constraints. In order to dispose of this at the end of the evaluation process, we also need a postprocessing step to delete the input material together with the additional symbols from the output tape of the OTt. This is simply an obligatory replacement rule mapping the unwanted material to the empty word.

⁸Square brackets indicate natural classes of phonemes, e.g. $[\text{CUnvoi}] = \{p, t, k, f, x, s\}$, which can also be achieved by the intersection of classes representing individual features: $[\text{Consonant}] \ \& \ [\text{Unvoiced}]$. They are distinctive subsets of the alphabet of the automaton.

Definition 12. *Postprocessing*

obligatory: (\([Phone]\\$ \mid \backslash)) \rightarrow [\langle \epsilon \rangle]

4.3. The EVAL Function as Regular Relation

According to the ranking of the five specified constraints, we need a dimension of 4 for the weighting semiring structure. We need only 4 dimensions, because two constraints are at the same level of the ranking, which means that they share a dimension in their weight vectors. The vector weights will be shown between angle brackets with their dimensions separated by ;. The individual constraints are modelled as obligatory replacement rules, that reweight the $\$$ in the given contexts. Since they are applied obligatorily, there has to be done at least one complementation during compilation in order to achieve robustness. This is unfortunately unavoidable and a grammar engineer has to be careful not to formulate constraints that increase the complexity of the whole system.

IDENT-IO(PLACE). This faithfulness constraint is introduced to restrict the output of GEN to a more reasonable set of candidates. This is also the reason why we rank this constraint highest in the constraint hierarchy. This constraint requires the identity of input-output-pairs in their articulation place feature and thus allows only for voice feature changes at a certain segment in the input. We model this by composing a set of transducers⁹ that each implement this constraint for a pair of phonemes differing only in their voice feature. We define $PoA = \{[Dental], [Velar], \dots\}$ as the set of natural classes of places of articulation. The ampersand symbolizes the intersection between two regular expressions, whereas \circ marks the composition.

$$\bigcirc_{\alpha \in PoA} \$ \rightarrow \$ \langle 1; 0; 0; 0 \rangle / [Obstruent] \& \alpha _ [Phone] - ([Obstruent] \& \alpha)$$

S-IDENT. This markedness constraint punishes candidates with adjacent obstruents that don't have the same value in their voice feature. We compose two expressions, one for each sequence of a voiced and an unvoiced consonant. The $/i$ operator permits the expression to ignore the appearance of a specified set of symbols, here any of the boundary symbols $\{\{, \}, \cdot\}$ (see [12] for the ignore operator).

$$\begin{aligned} \$ \rightarrow \$ \langle 0; 1; 0; 0 \rangle / _ ([CVoi] \backslash) \backslash ([Phone] \$ [CUnvoi] \backslash) / i [Boundary] \\ \$ \rightarrow \$ \langle 0; 1; 0; 0 \rangle / _ ([CUnvoi] \backslash) \backslash ([Phone] \$ [CVoi] \backslash) / i [Boundary] \end{aligned}$$

IDENT-PWOS. This faithfulness constraint ensures identity in the voice feature of stops between input and output segments in the onset of a prosodic word. Like above we achieve this by composing two expressions:

$$\begin{aligned} \{ (\backslash ([Phone] \$ [Consonant] \backslash)) * (\backslash ([CVoi] \& [Stop] \\ \$ \rightarrow \$ \langle 0; 1; 0; 0 \rangle / _ \\ [CUnvoi] \& [Stop] \end{aligned}$$

⁹The individual transducers are represented by Kaplan&Kay-style replacement rules, which have the general form of $\phi \rightarrow \psi / \lambda _ \rho$ (cf. [12, 16]). Its meaning can be stated by 'Replace ϕ by ψ in the left context λ and the right context ρ '. Every component $\phi, \psi, \lambda, \rho$ has to be a regular expression itself, where every component can possibly denote the empty word but only ψ can be weighted.

$$\begin{array}{c} \{ (\backslash ([\text{Phone}] \$ [\text{Consonant}] \backslash)) * \backslash ([\text{CUnvoi}] \& [\text{Stop}] \\ \$ \rightarrow \$ \langle 0; 1; 0; 0 \rangle / _ \\ [\text{CVoi}] \& [\text{Stop}] \end{array}$$

$\{ [+VOICE] \}_\omega$. This markedness constraint punishes candidates with voiced consonants that appear at the end of a prosodic word.¹⁰

$$\$ \rightarrow \$ \langle 0; 0; 1; 0 \rangle / _ [\text{CVoi}] \backslash \backslash \}$$

IDENT-IO. This faithfulness constraint requires identity between corresponding segments of input and output. We compose a set of transducers, which each reweight $\$$ in contexts of non-identical in- and output symbols.

$$\bigcirc_{\alpha \in [\text{Phone}]} \$ \rightarrow \$ \langle 0; 0; 0; 1 \rangle / \alpha _ ([\text{Phone}] - \alpha)$$

The EVAL function itself is then the composition of the outlined transducers representing the individual constraints of this OT system.

5. Conclusion

By using regular operations only, we have shown that Optimality Theory in general can be modelled by weighted finite-state algebra with the restriction, of course, that the constraints have to be expressible by regular relations. Particularly, the system can handle an arbitrary number of constraint violations, and we don't need purpose-built algorithms for evaluation and compilation of the system. Since the tropical vector semiring is a natural generalisation of the common tropical semiring, we can imagine that the same generalisation applied to other semirings could unlock new fields of application.

In our work, we have chosen an algebraic approach because it helped us to get a grip on the problem without getting lost in thousands of states and transitions. Therefore, we want to emphasize – in the tradition of Kaplan & Kay – the usefulness and hence importance of a higher-level approach to this class of phenomena that abstracts away from the specific automaton, focusing on the languages and their algebra.

¹⁰This constraint models the Dutch final devoicing as a side effect.

References

- [1] Robert Frank and Giorgio Satta. Optimality theory and the generative complexity of constraint violability. *Computational Linguistics*, 24(2):307–315, June 1998.
- [2] Lauri Karttunen. The proper treatment of optimality in computational phonology. In Kemal Oflazer and Lauri Karttunen, editors, *Finite State Methods in Natural Language Processing*, pages 1–12, Ankara, Turkey, July 1998. Bilkent University.
- [3] Jason Riggle. *Generation, recognition, and learning in finite state Optimality Theory*. PhD thesis, University of California, Los Angeles, 2004.
- [4] Alan Prince and Paul Smolensky. Optimality Theory: Constraint interaction in generative grammar. Manuscript, Rutgers University and University of Colorado at Boulder. Available at <http://roa.rutgers.edu/files/537-0802/537-0802-PRINCE-0-0.PDF>, 1993.
- [5] T. Mark Ellison. Phonological derivation in optimality theory. In *Proceedings of the 15th conference on Computational linguistics*, pages 1007–1013, Morristown, NJ, USA, 1994. Association for Computational Linguistics.
- [6] Jason Eisner. Efficient generation in primitive Optimality Theory. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 313–320, Madrid, July 1997.
- [7] Corinna Cortes and Mehryar Mohri. Context-free recognition with weighted automata. *Grammars*, 3(2/3):133–150, 2000.
- [8] Werner Kuich and Arto Salomaa. *Semirings, Automata, Languages*. Number 12 in EATCS Monographs on Theoretical Computer Science. Springer, 1988.
- [9] Géraldine Legendre, Yoshiro Miyata, and Paul Smolensky. Harmonic grammar: A formal multi-level connectionist theory of linguistic well-formedness: An application. Technical Report CU-CS-464-90, Department of Computer Science and Institute of Cognitive Science, University of Colorado, Boulder, 1990.
- [10] Jonas Kuhn. *Formal and Computational Aspects of Optimality-theoretic Syntax*. PhD thesis, Institut für maschinelle Sprachverarbeitung, Universität Stuttgart, 2001.
- [11] Joerg Didakowski. Syncop – combining syntactic tagging with chunking using weighted finite state transducers. In *Proceedings of Finite State Methods and Natural Language Processing (FSMNLP 07)*, 2007.
- [12] Ronald M. Kaplan and Martin Kay. Regular models of phonological rule systems. *Computational Linguistics*, 20(3):331–378, September 1994.
- [13] René Kager. *Optimality Theory*. Cambridge University Press, Cambridge, 1999.
- [14] Janet Grijzenhout and Martin Krämer. Final devoicing and voicing assimilation in dutch derivation and cliticization. Working Paper 106, SFB 282, Heinrich-Heine-Universität Düsseldorf, 1998.
- [15] Lauri Karttunen. Directed replacement. In Arivind Joshi and Martha Palmer, editors, *Proceedings of the Thirty-Fourth Annual Meeting of the Association for Computational Linguistics*, pages 108–115, San Francisco, 1996. Morgan Kaufmann Publishers.
- [16] Lauri Karttunen. The replace operator. In *Meeting of the Association for Computational Linguistics*, pages 16–23, 1995.

A Compression Method for Natural Language Automata

Lamia Tounsi, Béatrice Bouchou, Denis Maurel

Université François Rabelais Tours, LI

lamia.tounsi@computing.dcu.ie

{beatrice.bouchou, denis.maurel}@univ-tours.fr

Abstract. This paper deals with Finite State Automata used in Natural Language Processing to represent very large dictionaries. We present a method for an important operation applied to these automata, the compression with quick access. Our proposal is to factorize subautomata other than those representing common prefixes or suffixes. Our algorithm uses a DAWG of subautomata to iteratively choose the best substructure to factorize. The linear time accepting complexity is kept in the resulting compact automaton. Experiments performed on ten automata are reported.

Keywords. Automaton, Compression, DAWG, Greedy Algorithm, Electronic Dictionaries.

Introduction

A standard way to represent natural language dictionaries is the use of Finite State Automata FSA due to the high speed access (successful look up is performed in time proportional to the length of word and unsuccessful search is stopped as soon as there is no transition that continues the word). It is important to minimize the size of such huge data structures, while preserving their access time advantage. This task has been extensively studied. The most quite basic proposal has been to build a trie, where identical prefixes of different words are factorized. The trie can also be compressed [1]. Minimizing deterministic finite state automata leads to better space savings because identical suffixes are also factorized [2], [3], [4]. In addition, space savings can be achieved by optimizing the physical storage method as shown in [5].

In all of these approaches, identical parts that are neither prefix nor suffix still appear several times in the automaton. In [1] a quadratic method of trie compaction is presented, that involves coding the automaton so that not only common prefixes or suffixes are shared, but also common internal patterns. The procedure consists of a Ziv Lempel compression of a trie represented as a linked list.

The method proposed in this paper is linear, it is applied to deterministic FSA, which can be minimal or not. It proceeds in two main steps, which are independent from each other:

- A lossless compression at the internal data representation level.
- A greedy algorithm to factorize repeated identical parts inside the automaton ([6] uses close method for large strings, as DNA...).

This algorithm iteratively chooses the best factorization to reduce the size of the original automaton. The heuristic used for this choice is based on information computed from the automaton and stored in a data structure similar to a Directed Acyclic Word Graph DAWG¹. At the beginning of the algorithm, we apply to the original automaton a reducing process similar to the one presented in [10] in order to detect all its sequential or parallel subautomata; and we store these subautomata in the DAWG structure. At each iteration step, the DAWG indicates the best candidate subautomaton to factorize. Then, factorization is performed in the automaton and the DAWG structure is incrementally updated.

Notice that the DAWG structure used for factorization can be seen as an index of sequences and parallels occurring in the automaton. In the same way as automatic indexation of text draws up a list of all its words (each word with all its positions), this kind of DAWG represents a list of substructures with their positions inside the automaton. Information such as type (sequence or parallel), size, frequency and number of transitions are also stored in this DAWG.

The paper is organized as follows: we first give the intuition of the lossless internal representation compression on the one hand, and of the sequences and parallels' recognition on the other hand. In Section 2 we present the adapted DAWG structure, and in Section 3 we show how it is used for factorization. We analyze in Section 4 the experimental results of the compression-factorization process and the accessibility features on the resulting compressed structure.

1. Internal representation, sequences and parallels

In this paper, we deal with (possibly minimized) acyclic deterministic finite automata with an end marker $A = \langle \Sigma, Q, \delta, q_i, q_f \rangle$, where Σ is the alphabet, Q is the finite set of states, δ is the transition function ($\delta: Q \times \Sigma \rightarrow Q$), q_i is the initial state ($q_i \in Q$) and q_f is the final state ($q_f \in Q$).

An automaton A is represented in memory by the list of its output transitions, each transition t being associated with three data: (i) a binary flag indicating whether t is the first transition of the current state, (ii) the label of t and (iii) the address of the target state. In Figure 1 we show such a list of transitions and the finite automaton it represents ($\#$ is the end character). Notice that transitions are organized according to the height of their initial state q (the size of the longest path linking q to q_f).

We apply a coding of transitions using a constant size unit, as follows:

(i) *binary flag*: 1 bit; (ii) *label*: the minimum number of bits necessary to encode the alphabet, $Size_{alphabet} = \log_2(|\Sigma|)$; (iii) *address of target state*: the minimum number of bits necessary to encode the biggest value $ValMax$ used as target state identifier, $Size_{address} = \log_2(ValMax + 1)$.

Variable size compression techniques would have not been efficient in our context because data are numerous and different from each others, which would lead to a correspondence table that is too large in the compressed files. We have checked this result by applying Huffman coding. We also choose absolute addressing because a dictionary contains short words that require big address jumps, so relative addressing would not be interesting.

¹A special form of minimal deterministic automaton built to recognize and analyze repetitions in a text, or to develop a complete index over a text [7], [8], [9].

We can easily rebuild the initial automaton from the compressed representation due to the following information (these information are added as a header to the compressed file): (i) number of transitions, (ii) $Size_{alphabet}$ and (iii) $Size_{address}$. This operation is not so simple for other compressions (such as LZW [1]).

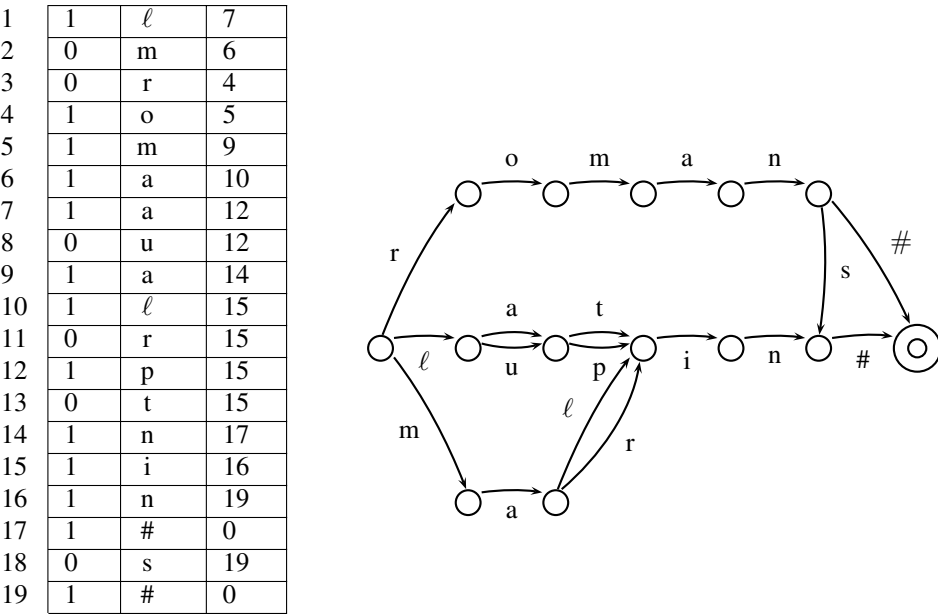


Figure 1. Initial automaton

This coding method is close to the first one used by Daciuk [5]. He also uses a function that returns the number of bytes needed to store all the arguments in order to compute the size of the automaton in bits.

Our experiments were carried out on several automata representing dictionaries, the average compression ratio for the tested automata is around 72%-74%, see Table 3. For instance, the size of the french automaton in text format is 2 178 Ko, this size decreases to 563 Ko using our compression method (binary encoding).

The other way to decrease the size of the automaton is to reduce its number of states and transitions, by factorizing repeated internal substructures. To this aim, we have first developed algorithms to detect substructures of any shape [11] and our experimental results have shown that, in natural language dictionaries, the frequent substructures are almost transition sequences.

An efficient way to detect all sequences in an automaton is to apply the reduction process presented in [12], whose principles are as follows :

- while there is a transitions sequence or a set of parallel transitions between two states,
- (i) replace each longest transition sequence with one transition having a new label;
- (ii) replace each set of parallel transitions between two states with one transition having a new label.

In other words, the sequences are jammed (resp. parallel are flattened) by one transition with a new label. New labels correspond to symbols in an extended alphabet, that must be stored in memory. Figure 2 shows the reduced automaton of Figure 1 and its extended alphabet. Figure’s 2 table includes the regular expressions for the external alphabet symbol (labels), e.g. "13 parallel 5 6" is "a|u" and "sequence 10 9" is "i.n".

Obviously, reducing all substructures as in our example is not interesting for space saving. Indeed, the factorization of a sequence or a parallel occurring once extends the alphabet, without reducing any occurrence. Thus it increases the storage size. In order to reduce the storage size, it is essential to choose the right substructures to factorize: this can be achieved if we can compare features related to substructures, such as their size and their frequency.

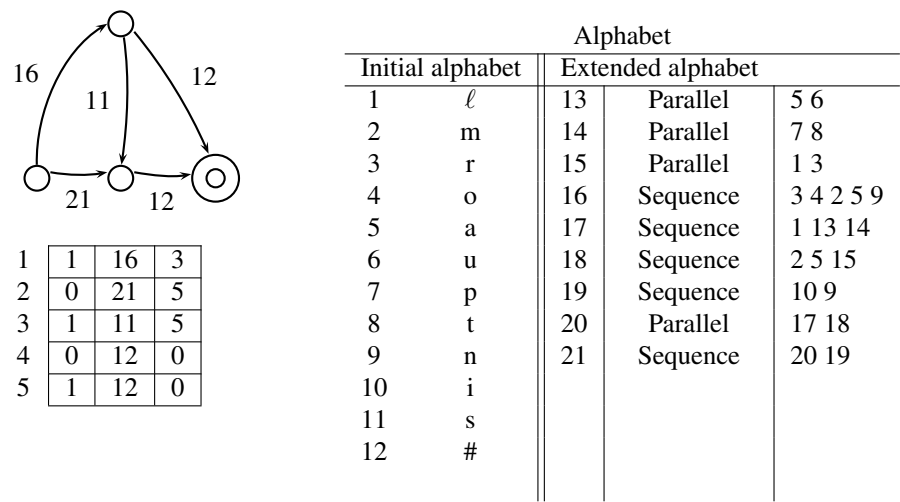


Figure 2. Reduced Automaton

An efficient way to get this information is to build the Directed Acyclic Word Graph (*DAWG*) of all sequences (and parallels) found. Indeed, the *DAWG* is a (space-efficient) data structure to treat and analyze repetitions in a text [7], [8].

2. Indexation

Precisely, the *DAWG* is the smallest finite state automaton that recognizes all suffixes of a given string [7], [13], [8]. It is constructed in linear time with respect to the size of the given string. This structure allows to compute the number of subwords in a word, the longest repeated subword, etc. In [9], Mehryar Mohri adds new information to the *DAWG* in order to use it as an index. More precisely, he associates with each state *p* of the *DAWG* a list of integers representing the positions of the words recognized at *p*. So, when a word is red in the *DAWG*, all its positions in the input text can be obtained directly from the list of num-

bers associated with the reached state.

In our context, as we want to factorize common substructures in a given automaton, it is useful to keep in the *DAWG* their positions in the automaton. That is why we start from Mohri's proposal and we adapt it. The *DAWG* is built by considering sequences and parallels (found in the automaton) as words.

The following steps describe the adaptation for sequences :

1. *Convert input data:* Consecutive sequences of transitions detected in the automaton are converted into words by sequencing consecutive transition labels (in the order of transitions).
2. *Do not consider final state mark:* In our context each state of the *DAWG*, final or not, is interesting because it recognizes a list of suffix and/or infix of the original sequence, so, we don't use a mark to denote if a state is final or not.
3. *Use double index:* we associate with p a double index to represent the positions (i) in the automaton and (ii) in the sequence itself.

In addition, the adaptation for parallels is as follows :

All parallel transitions between two nodes are converted into words by sequencing transitions labels in a given order. Remember that the purpose is to detect all identical parallels in the automaton. For this reason, an additional processing is necessary to generate all subparallels of these "parallel" words: for each "parallel" word, we must involve 2^{n-2} words, where n is the number of transitions in parallel.

For instance for the "parallel" word "abcde", we build a *DAWG* from the following list: "ae", "abe", "ace", "ade", "abce", "abde", "acde" and "abcde".

Example 1

Let $S_1 = ccdbaab$, $S_2 = baab$ and $S_3 = ccdb$ be 3 sequences detected in an automaton A . Assume that S_1 occurs 4 times in A , S_2 appears once more in A (i.e. outside S_1) and S_3 happens twice more in A . In other words S_2 appears 5 times in A and S_3 6 times. Figure 3 presents the *DAWG* of S_1 , S_2 and S_3 .

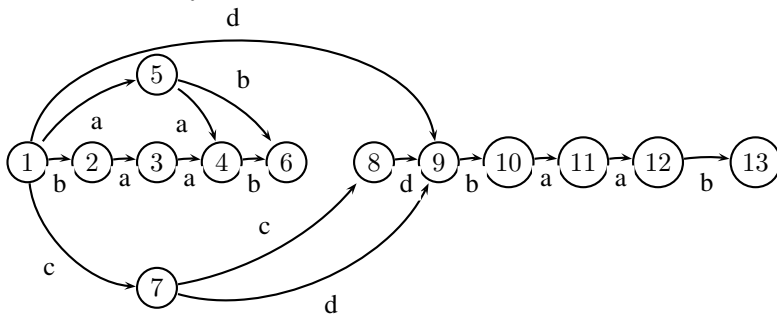


Figure 3. $DAWG(S_1, S_2, S_3)$

Each state p of the DAWG in Figure 3 is associated to an index representing the positions of all subsequences of S_1 , S_2 and S_3 recognized at p . This index indicates also the frequency of the subsequences.

Indexes are as follows:

$Index(1) = \emptyset$	$Index(8) = S_1\{2\}, S_3\{2\}$
$Index(2) = S_1\{4, 7\}, S_2\{1, 4\}, S_3\{3\}$	$Index(9) = S_1\{3\}, S_3\{3\}$
$Index(3) = S_1\{5\}, S_2\{2\}$	$Index(10) = S_1\{4\}, S_3\{4\}$
$Index(4) = S_1\{6\}, S_2\{3\}$	$Index(11) = S_1\{5\}$
$Index(5) = S_1\{5, 6\}, S_2\{2, 3\}$	$Index(12) = S_1\{6\}$
$Index(6) = S_1\{7\}, S_2\{4\}$	$Index(13) = S_1\{7\}$
$Index(7) = S_1\{1, 2\}, S_3\{1, 2\}$	

For instance, the state 4 recognizes two subsequences 'aa' and 'baa' and its index shows that these subsequences occur once in S_1 and once in S_2 . The subsequence 'aa' appears in position 5 inside S_1 ($6 - |aa| + 1$) and in position 2 inside S_2 ($3 - |aa| + 1$); the subsequence 'baa' appears in position 4 inside S_1 ($6 - |baa| + 1$) and at the beginning of S_2 ($3 - |baa| + 1$).

3. Factorization

In order to reduce the size of storage, it is essential to choose the best substructures to factorize. Our process uses a function called *profit* and noted Δ , to compute for each substructure i) the memory space saved by removing all its occurrences from the original automaton and ii) the memory consumed by extending the size of the alphabet to represent it.

The difference between these two sizes indicates whether it is advantageous to reduce this substructure or not. So, the best substructures to factorize is the one that maximizes Δ .

Example 2

Consider again Example 1. Assume that the alphabet can be extended to a maximum of 8 characters and the automaton A contains less than 128 transitions. Thus, 11 bits are necessary to represent one transition (binary flag: 1 bit, label: 3 bits and target address: 7 bits).

If we factorize a substructure of L transitions and frequency F then we save $F * (L - 1) * 11$ bits ($L - 1$ because we replace the substructure by one transition in A).

In this example, the size of a substructure is necessarily less than 8 (S_3 is the longest substructure and $|S_3|=7$), so 3 bits are necessary to store it and $3 * L$ bits are used for the list of labels. Finally $3 + 3L$ bits are consumed, so:

$$Profit = F * (L - 1) * 11 - 3 - 3L \text{ bits}$$

- If we factorize 4 S_1 and 2 S_3 in A : Profit = 291 bits.
- If we factorize 5 S_2 and 6 ccd in A : Profit = 292 bits.
- If we factorize 5 S_2 and 4 ccd and 2 S_3 in A : Profit = 277 bits.
- If we factorize 5 aab and 6 S_3 in A : Profit = 183 bits.

Several scenarios are possible and each one leads to a different profit. Indeed, with each state p in the *DAWG* is associated one or more substructures with their associated profit. As we aim to have a light process, we apply a first heuristic to decrease the number of possibilities which consists of giving the advantage to the longest substructures. So, with each state p in the *DAWG* we associate the profit of the factorization of its longest substructure. The size of the longest substructure at p is given by the depth of p .

Our approach uses a greedy algorithm to compress an automaton : the general idea is to make at each step the best local choice, without backtracking, with the hope of finding the best global compression.

Algorithm 1 computes the set of substructures chosen for compression, noted *CSA*. For each element of *CSA* it uses a unique label α (from the extended alphabet) to represent it in A . Algorithm 1 starts by running a depth first exploration of A to compute the set of substructures (sequence or parallel transitions), noted *SA*. Then, it builds the *DAWG* of *SA*, computes Δ and selects the state q that maximizes Δ to compress its longest subsequence. More precisely, at each iteration, the chosen subsequence is removed from the *DAWG*, replaced by α in A and *SA*, and new subsequences containing α are added in the *DAWG*. Algorithm 1 stops when there is no more attractive state in the *DAWG* or when the alphabet is full (no more available new label).

Algorithm 1 Compression

input/output: A - output: *CSA*

```

Let  $SA$  be the set of subautomata (sequences or parallels) :  $SA \leftarrow \emptyset$ ;
Let  $CSA$  be the set of factorized substructures :  $CSA \leftarrow \emptyset$ ;
Compute the set  $SA$  from  $A$  and generate DAWG from  $SA$ ;
Select from DAWG a state  $q$  that gives the most attractive space saving; /*using  $\Delta$ */
while DAWG contains at least one attractive  $q$  and the alphabet is not full do
    Let  $w$  be the longest path in DAWG ending at  $q$  and let  $\alpha$  be a new character;
    Add  $w$  to CSA;
    Jam  $w$  with  $\alpha$  in  $A$ ;
    Update DAWG and  $SA$ ; /* add new elements containing  $\alpha$ */
    Select from DAWG a state  $q$  that gives the most attractive space savings;
end while

```

To factorize a substructure w updates the initial automaton and the *DAWG*. When w is removed from the *DAWG* we also remove all indexes that refer to w . We delete a state p from the *DAWG* when its index becomes empty.

If w is included in another longer sequence w' , then its factorization induces to create a new sequence where w is replaced by α in w' . More precisely, the proposed algorithm performs optimal insertion to a minimal *DAWG*, which means that after any insertion the *DAWG* remains minimal. The time required to add a new sequence is $O(n)$ with respect

to the size of the DAWG. Repetitive insertions construct minimal deterministic DAWG incrementally and each sequence insertion traverses only a limited portion of the graph. Finally no additional minimization operation is required [14].

4. Experimental Results

4.1. Experiments on compression

We have implemented our algorithms in C language, using a PC with a Pentium 4, 4.80GHZ and 512 MB of RAM. We have performed tests on ten FSA representing NLP dictionaries. These dictionaries are very large so the size of these automata remains large, in spite of the minimization.

We distinguish two kinds of dictionaries :

- Category 1 : the entries of these dictionaries are formatted using the DELA formalism [15]
- Category 2: the entries of these dictionaries are collected from the web (newspapers) and contain numbers and special characters such as the @ ? ! . , etc.

Table 3 presents a selection of relevant values concerning experiments conducted on these automata once minimized: for instance the dictionary of French (DELAF Fr) contains 637 282 words and uses 71 characters, its automaton has 67 995 states and 177 465 transitions. The size of this automaton in text format is 2 178 KB.

Using the binary encoding allows to decrease the size to 563 KB. Using our compression algorithm reduces this size to 529 KB while extending the alphabet to 128 characters.

	Dictionary		Initial automaton			Compressed automaton		
	Words	Initial alphabet	Transitions	States	Initial size(KB)	binary encoding (KB)	Compression (KB)	Size alphabet
Category 1								
DELAF Fr	637 282	71	177 465	67 995	2 178	563	529	128
DELAF En	282 338	47	252 664	116 848	3 081	801	735	256
DELAF Sr	1 214 417	52	193 668	61 383	2 340	591	585	128
DELAF De	3 713 121	89	335 284	14 2795	4 133	1 105	1 036	256
Cities Fr	35 391	75	95 589	61240	1108	291	222	2 048
Poly En	320 424	31	717 112	435940	8 963	2451	1 761	1 024
Category 2								
Web Fr	236 057	43	298 117	101 837	3 560	946	935	128
Web Bg	191 738	43	270 495	113 678	2 503	638	610	128
Web Hu	165 073	42	209 209	85 661	3 235	858	757	128
Web Pt	398 839	49	538 697	214 992	6 510	1 776	1 653	128

Table 3. Automata compression

Table 3 shows that on average, the compression algorithm reduces the size of an automaton by 75% compared to its size in text format and it reduces the size of an automaton by 10% compared to its size when the binary encoding is applied.

We note that, in general, the most efficient factorizations use short alphabet (7 to 8 bits). However, the dictionary of polylexical words in english reaches its best compression ratio using 10 bits.

The dictionary of French city names needs 11 bits to reach its best compression ratio.

Indeed, the compression algorithm reduces the size by 80% compared to the automaton in text format and by 24% compared to the automaton only binary coded. This means that the amount of repeated strings that are neither prefixes nor suffixes is high for this automaton.

The choice of substructures to factorize depends on the size and the frequency of these substructures inside the automaton, but may be also on their position occurrences. The heuristic used may be changed: for instance it can be interesting not to factorize the longest sequence recognized at a state of the *DAWG*, or not to factorize all its occurrences inside the automaton in order to be able to anticipate some possible combinations of factorization.

Table 3 is a selection of results. We have performed numerous tests by varying parameters such as the size of alphabet, the order of coding and factorizing, or different format of input data, for instance:

- i) non-minimal automaton,
- ii) dictionary in text format,
- iii) for further experimentation, we have reversed the words of dictionaries and we have processed again the algorithm for the three variants (minimal automaton, non-minimal automaton, dictionary in text format).

The results proved that compression is not necessarily more efficient when it is applied to formats different from a minimal FSA. Applying Algorithm 1 to non-minimal automata is efficient for automata of category 2 (Web Fr, Bg and Pt) and processing Algorithm 1 directly to dictionaries in text format (lexicon) is suitable for the two following dictionaries:

- Web Pt : first, we have applied Algorithm 1 to the lexicon of web Pt, then we have built the minimal automaton of the compressed lexicon, the resulting size is 1 600 KB (3% better than the size presented in Table 3).
- French cities: we have performed the same process and we have got a resulting size equal to 67 KB; this method improves by 70% the size of the compressed French cities automaton presented in Table 3 (222 KB). We have analyzed the factorized substructures and we have detected many common substrings in the names of French cities such as *Saint, Bourg*, etc.

Also, reversing words is not suitable for automata compression with one exception for the automaton of polylexical English whose size is 3% smaller than the size presented in Table 3 with this method.

These results can be compared to those presented in [5], where several compression methods for finite states automata representing morphological dictionaries are analyzed. The author uses pointers and tables to represent these automata, and he applies various compression techniques to code data, such as sharing the same space for some parts of the automaton, eliminating transitions counters, changing the order of some transitions, etc. In particular, our experiments lead to the same conclusion: there is no one efficient method for automata compression, it depends highly on the input data.

In [1] the authors represent and store natural language data using tries. Once a trie is generated, it is transformed into a linked list before being compressed by the method of

Ziv and Lempel. The search of similar structures uses tree suffixes (or a table of suffixes). In comparison with our work, the compression ratio obtained on DELAF French is better because they recognize and merge more substructures than us. On the other hand, their method is quadratic while our process is linear.

4.2. Management of Compressed Automata

Our compression method addresses a special case of the problem presented by Kalin Georgiev in [16], in that paper, he states that such a factorization preserves *FSA* structure. Moreover, this compression keeps an automaton's properties valid:

Our data structure does not change, our compression preserves *FSA* structure. Indeed, we factorize only the substructures that can be replaced by one transition in the original automaton. The compressed automaton is also deterministic, minimal, etc. if the original automaton is. Notice also that our method allows compression only if the language recognized does not change. Consequently, we can safely and easily use and manage compressed automata: read, write, update, etc.

Let A be a compressed automaton and w a word. Note that the alphabet of A is composed of i) initial alphabet which contains all the initial characters and ii) extended alphabet which lists all the chosen factorized substructures.

As A is deterministic, we can check if w belongs to A by finding a successful path through the compressed automaton (from q_i to q_f). For each step, we compare labels of outgoing transitions from the current state, noted p , with the current character of w , noted w_i . We cross immediately p if w_i belongs to its initial alphabet else we check if w_i is the first character of a sequence or is included in a parallel structure of the extended alphabet.

More precisely, i) when w_i is the first character of a sequence of length ℓ , then $w_i \cdots w_{i+\ell-1}$ must exactly correspond to the characters of the sequence. ii) As parallels are represented in memory as sequences, when w_i is a character of a parallel structure, we should check it with all the others characters. To speed up the search, it is helpful to order the characters of a parallel (for example, using the alphabetic order).

The same process is applied when a sequence includes another sequence, or a parallel.

Compressed automaton vs. uncompressed automaton

We have compared the time elapsed for rebuilding the French dictionary (637 283 words) i) from an automaton compressed by Algorithm 1 and ii) from an initial automaton. The results show that the time necessary to rebuild all words from the original automaton is 0.7s and it takes 0.1s more to do it from the compressed automaton.

Updating compressed automaton

The following incremental updates are possible in a compressed automaton A :

- Delete a word $w \in L(A)$:

Assume that w induces the path $q_i, q_1, \dots, q_n, q_f$ in A . The three following cases are possible :

1. q_1, \dots, q_n are not divergent² and not convergent, so deleting w removes all the states q_1, \dots, q_n from A .
2. q_1, \dots, q_n are not convergent but can be divergent, so deleting w removes all the successive states starting from the last divergent state.
3. q_1, \dots, q_n are convergent (at least one state is convergent), so to delete only w , and not the other words that share the path q_1, \dots, q_n with w inside the automaton, we first deminimize the automaton for that path q_1, \dots, q_n (make it non-minimal), then we can delete w as case 1 or case 2.

- Add a word $w \notin L(A)$:

This insertion must keep A deterministic, the two following cases are possible:

1. when any outgoing transition of q_i is labeled with the first character of w (initial or extended alphabet), then a new path composed of not divergent and not convergent states is added to A using its initial or extended alphabet.
2. when a starter segment of w is recognized in A and ends at a state $p \in A$, then i) this segment is deminimized in A using the initial alphabet then ii) A is completed from p by a sequence of not convergent and not divergent states to represent the rest of w .

As presented, we can easily add or delete a word from a compressed automaton but these updates can't always preserve the compression, for instance, the addition of a list of words can reveal new substructures to reduce, in this case, executing again Algorithm 1 can be useful.

Conclusion

In this paper, we have presented a compression algorithm for automata that represent dictionaries, while keeping their fast access advantage. The linear time accepting complexity is kept in the resulting compact automaton. This algorithm iteratively chooses the best factorization to reduce the size of the original automaton. The heuristic used for this choice is based on information stored in a DAWG. This DAWG is built from automaton's substructures, more precisely from sequences and parallel transitions. These substructures are discovered using a reducing process similar to the one in [12] and [10].

As shown in the experimental results, our proposal is adaptable for each automaton. The parameters to adapt are those of the internal coding, such as the size of alphabet, or the structure of the input automaton, or the heuristic used to choose which substructure to factorize at each step. Moreover, we have considered sequences (and parallel) transitions because they are the most frequent in automata representing dictionaries. Future work is to extend our proposal for any kind of substructure, provided that it can be replaced by one transition.

²A divergent (resp. convergent) state is a state with more than one outgoing (resp. incoming) transition.

Acknowledgments

The authors thank Prof. Franz Guenther, Prof. Éric Laporte, Prof. Jacques Savoy and Prof. Duško Vitas for free use of their dictionaries.

References

- [1] S. Ristov and E. Laporte. Ziv Lempel compression of huge natural language data tries using suffix arrays. *Journal of Discrete Algorithms*, pages 241–256, 1999.
- [2] S. Mihov. Direct construction of minimal acyclic finite states automata. *Annuaire de l'Université de Sofia St. Kl. Ohridski*, 91(1), 1998.
- [3] J. Daciuk. *Incremental Construction of Finite-State Automata and Transducers, and their Use in the Natural Language Processing*. PhD thesis, Technical University of Gdansk, Poland, 1998.
- [4] B. W. Watson. A new algorithm for the construction of minimal acyclic dfas. *Science of Computer Programming*, 48(2–3):81–97, 2003.
- [5] J. Daciuk. Experiments with automata compression. In *CIAA 2000*, volume 2088 of *LNCS*, pages 105–112, 2000.
- [6] A. Apostolico and S. Lonardi. Off-line compression by greedy textual substitution. *Proc. IEEE*, 88(11):1733–1744, 2000.
- [7] A. Blumer, J. Blumer, D. Haussler, A. Ehrenfeucht, M. Chen, and J. Seiferas. The smallest automaton recognizing the subwords of a text. *Theoretical Computer Science*, 40:31–55, 1985.
- [8] M. Crochemore and R. Verin. On compact directed acyclic word graphs. In J. Mycielski, G. Rozenberg, and A. Salomaa, editors, *Structures in Logic and Computer Science*, LNCS 1261, pages 192–211. Springer-Verlag, 1997.
- [9] M. Mohri. On some applications of finite-state automata theory to natural language processing. *Natural Language Engineering*, 2:1–20, 1996.
- [10] N.D. Beijer, B. Watson, and D. Kourie. Stretching and jamming of automata. In *RSA 2003 SAICSIT*, pages 198–207, 2003.
- [11] L. Tounsi. *Sous-automates à nombre fini d'états. Application à la compression de dictionnaires électroniques*. PhD thesis, Université François Rabelais Tours, France, 2007.
- [12] J. Valdes, R.A. Tarjan, and E.L. Lawler. The recognition of series parallel digraphs. *SIAM J. Comput.*, 11(2):298–313, 1982.
- [13] A. Blumer, J. Blumer, D. Haussler, R.M. McConnell, and A. Ehrenfeucht. Complete inverted files for efficient text retrieval and analysis. *J. ACM*, 34(3):578–595, 1987.
- [14] K. Sgarbas, N. Fakotakis, and G. Kokkinakis. Optimal insertion in deterministic DAWGs. *Theoretical Computer Science*, 301:103–117, 2003.
- [15] B. Courtois and M. Silberztein. Dictionnaires électroniques du français. *Langues française*, 87:11–22, 1990.
- [16] K. Georgiev. Compression of minimal acyclic deterministic FSAs preserving the linear accepting complexity. In *Workshop FSTAS 2007*, pages 7–13, Bulgaria, 2007.

Event Extraction for Italian Using a Cascade of Finite-State Grammars

Vanni ZAVARELLA, Hristo TANEV¹ and Jakub PISKORSKI

Joint Research Centre of the European Commission, Ispra, Italy

Abstract. This paper reports on our experience of adapting a real-world live event extraction system based on a cascade of finite-state extraction grammars to the processing of a new language, namely Italian. The real-time event extraction processing chain and the pattern specification language are briefly presented. The major part of the paper focuses on the creation of event extraction grammars and related resources for English and their adaptation for extracting events in Italian news articles. Some interesting phenomena which complicate the event extraction task for Italian are pinpointed and the results of the evaluation are presented. In particular, we compared two versions of the system for Italian, one based on surface-level patterns and a hybrid one, which integrates slightly more linguistically sophisticated patterns for covering a rich variety of morphological and syntactic constructions in Italian.

Keywords. event extraction, finite-state grammars, Italian

Introduction

The task of event extraction is to automatically identify events in free text and to derive detailed information about them, ideally identifying *Who did what to whom, when, with what methods, where and possibly why*. Automatically extracting events is a higher-level information extraction (IE) task which is not trivial due to the complexity of natural language and due to the fact that a full event description is usually scattered over several sentences and articles. Although a considerable amount of work on automatic extraction of events has been reported [1,2,3], it still appears to be a lesser studied area in comparison to the somewhat easier tasks of named-entity and relation extraction. The research in this area was pushed forward by the Message Understanding Conferences² and by the ACE³ (Automatic Content Extraction) program.

This paper reports on our experience of adapting an existing real-world live event extraction system based on a cascade of finite-state grammars, to the processing of a new language, namely Italian. This system has been developed for automatically detecting violent and natural disaster events from online news. It has been integrated in Europe Media Monitor (EMM) [4], a web-based news aggregation system that collects about 50000 news articles from 1500 sources in 43 languages each day. There are certain require-

¹Corresponding Author: Hristo Tanev, e-mail: Hristo.Tanev@jrc.it

²<http://www.itl.nist.gov/iaui/894.02/related/projects/muc>

³<http://projects ldc.upenn.edu/ace>

ments for a real-time event extraction engine imposed by such a large-scale news aggregation engine, which have to be met in order to guarantee robustness and a fast linguistic resource development cycle. In particular, the following requirements are paramount:

- (a) efficient processing of large amount of news articles on a daily basis
- (b) ease in maintenance of existing resources by non-linguists and non-experts
- (c) extensibility to new languages within a short time period.

In order to meet the first requirement, an efficient IE-oriented grammar formalism has been developed. As for the two others, a linguistically lightweight approach has been applied, which heavily utilizes clustered news. In particular, only a small fraction of each text is processed, and solely simple, easily maintainable and to a large extent language-independent linear patterns are used instead of grammars involving more linguistic sophistication as reported elsewhere, e.g., in [5]. The main focus of the paper is on the development of event extraction grammars and related resources for English and their adaptation and modification for Italian. Some interesting phenomena of Italian which complicate the event extraction task are pinpointed. To our best knowledge this is the first attempt towards event extraction for Italian from online news.

This paper is structured as follows. First, in Section 1 the real-time event extraction processing chain is described. Section 2 introduces the pattern specification formalism. Next, Section 3 addresses the development of the event extraction grammar for English. Subsequently, Section 4 elaborates on adapting existing resources to extract events from Italian news. The results of the evaluation are presented in Section 5. Finally, a summary is given in Section 6.

1. Real-Time Event Extraction Process

First, news articles are gathered by dedicated software for electronic media monitoring, namely the EMM system [4], which regularly (every 10 minutes) checks for updates of news articles across multiple sites. Secondly, the input data is geo-tagged, categorized and grouped into news clusters, ideally including documents on one topic. For each such cluster the core event extraction engine tries to detect only the main event, analyzing all documents in the cluster, and to produce a frame, whose main slots are: date and location, number of killed, injured and kidnapped people, actors, and type of event. Initially, each article in a given cluster is linguistically pre-processed in order to produce a more abstract representation of the text. This encompasses: fine-grained tokenization, sentence splitting, morphological analysis and domain-specific lexicon lookup.

Once texts are grouped into clusters and linguistically preprocessed, a cascade of simple finite-state grammars is applied on each article within a cluster for detection of small-scale structures (e.g., unnamed and named person groups, numbers, person names, etc.) and for assembling them into partial event descriptions. The extraction grammars at each level are matched only against the first sentence and the title of each article. By processing only the top sentence and the title, the system is more likely to capture facts about the most important event in the cluster and at the same time it avoids capturing some prior events, which are frequently mentioned in news articles in the context of the main event. Since the information about events is often scattered over different articles, the final step consists of performing cross-article cluster-level information fusion, i.e.,

we aggregate and validate information extracted locally from each single article in the same cluster. In particular, victim counting, semantic role disambiguation and event type classification are performed at this stage of processing.

The event extraction engine is triggered every 10 minutes on a clustering system that has a 4-hour sliding window in order to keep up-to-date with most recent events. The subsequent sections focus on the pattern specification language and creating extraction grammars. For the details on news data gathering, clustering, geo-tagging, information fusion and visualization please refer to [6,7]. The linguistic preprocessing tools are described in [8].

2. Extraction Pattern Specification Language

In order to guarantee vast amount of textual data to be processed in real time, EXPRESS [9], an efficient extraction pattern engine has been developed. A grammar in the underlying formalism consists of so called pattern-action rules. The left-hand side (LHS) of a rule (the recognition part) is a regular expression over flat feature structures (FFS), i.e., non-recursive typed feature structures (TFS) without structure sharing, where features are string-valued and types are not ordered in a hierarchy like in unification-based grammars. The right-hand side (RHS) of a rule (action part) constitutes a list of FFS, which is returned in case the LHS pattern is matched. On the LHS of a rule variables can be tailored to the string-valued attributes in order to facilitate information transport into the RHS, etc. Further, functional operators (FO) are allowed on the RHSs for: (a) forming slot values, (b) establishing contact with the ‘outer world’, and (c) specifying boolean-valued predicates.⁴ Rules can be associated with multiple actions, i.e., producing multiple annotations (possibly nested) for a given text fragment. Finally, grammars can be cascaded and arbitrary processing resources can be integrated at any level of the cascade. The following pattern for matching events, where person groups are injured, illustrates the syntax.

```
injury-event :- ((person-group & [NAME: #name1, NUMBER: #num1]):injured1
    token & [SURFACE: "and"]
    (person-group & [NAME: #name2, NUMBER: #num2]):injured2
    injured-phrase & [FORM: "passive"]
):event
-> injured1: victim & [NAME: #name1, NUMBER: #num1],
    injured2: victim & [NAME: #name2, NUMBER: #num2],
    event: injury & [VICTIM: #name, NUMBER: #count],
    & #name = Concatenate(#name1, "and", #name2)
    & #count = EstimateNumber(#num1, " ", #num2).
```

This pattern matches a sequence consisting of: a person group (structure of type *person-group*), a conjunction ‘and’, another person group (the victim), followed by a phrase in passive form (structure of type *injured-phrase*), which triggers an *injury event*.⁵ The symbol & links a name of the FFS’s type with a list of constraints (in the form of attribute-value pairs) which have to be fulfilled. The variables #name1 and #name2 establish bindings to the names of both person groups involved in the event. Analogously, #num1 and #num2 establish bindings to the numbers of persons involved. Further, there are three labels on the LHS (*injured1*, *injured2*, and *event*) which

⁴New FOs can be added through implementing an appropriate programming interface.

⁵This pattern would match the text ‘Five terrorists and two civilians were wounded.’

specify the start/end position of the annotation actions specified on the RHS. The first two actions on the RHS produce two FFSs of the type `victim`, where the value of `NAME` and `NUMBER` slots are created through accessing the variables `#name1`, `#name2`, `#num1` and `#num2` on the LHS resp. Finally, the third action (`event`) produces an FFS of type `injury`. The value of the `VICTIM` attribute is computed via a call to a FO `Concatenate()`, which simply concatenates its arguments, whereas the value of the `NUMBER` slot is computed by ‘normalization’ of number expressions tailored to `#num1` and `#num2` resp. and summing them up (a call to `EstimateNumber()`).

Although the pattern specification language might appear less powerful than other IE-oriented formalisms, e.g., XTDL [10] or JAPE [11], it has two major advantages over the others: (a) grammar rules are easier to write by non-linguists (e.g., unification turns to be less intuitive for them) and non-programmers (e.g., JAPE allows code to be written on RHS of rules), and (b) grammars can be processed significantly faster. An overview of the formalism, grammar compilation and run-time performance comparison with other formalisms is given in [9].

3. Event Extraction Grammars for English

There are two different approaches to building event extraction grammars: (a) the complexity of the language is represented at the level of lexical descriptions where each word in the lexicon is provided with a rich set of semantic and syntactic features ([5,12]), (b) the complexity of the language is represented through different, mostly linear, patterns in the grammar, which rely on superficial or less sophisticated linguistic features [13].

Providing rich lexical descriptions like verb sub-categorization frames in [5] or ontologies in [12] requires a linguistic expertise, on the other hand more shallow lexical descriptions will result in more patterns to encode the necessary linguistic knowledge. However, superficial patterns are closer to the text data. We believe that their creation is more intuitive and easier for non-experts than building ontologies or sub-categorization frames. Writing such patterns is easier for languages like English with strict word-order and simple morphology. Therefore, we followed this approach for creating the English event extraction grammar.

This grammar in its current version consists of two subgrammars. The first-level subgrammar contains patterns for recognition of named entities (e.g., person names), numbers, quantifiers, simple chunks representing unnamed (e.g., *five civilians*) and named person groups (e.g., *thousands of Iraqis*). As an example consider the following rule for detecting mentions of person groups.

```
person-group :- ((gazetteer & [GTYPE: "numeral", SURFACE: #quant, AMOUNT: #num])
                 (gazetteer & [GTYPE: "person-modifier"]))?
                 (gazetteer & [GTYPE: "person-group-proper-noun", SURFACE: #name1])
                 (gazetteer & [GTYPE: "person-group-proper-noun", SURFACE: #name2])):name
-> name: person-group & [QUANTIFIER: #quant, AMOUNT: #num,
                        TYPE: "UNNAMED", NAME: #name],
   & #name = Concatenate(#name1,#name2).
```

This rule matches noun phrases (NP), which refer to people by mentioning their nationalities, religion or political group they belong to, e.g. *three young Chinese, one Iraqi Muslim*. The words and phrases which fall in the category `person-group-proper-noun`, `numeral` and `person-modifier` (e.g. *young*) are listed in a domain-specific lexicon (`gazetteer`). In this way the grammar rules are being kept language independent, i.e.,

rules can be applied to languages other than English, provided that language-specific dictionaries back up the grammar. Through abstracting from surface forms in the rules themselves the size of the grammars can be kept relatively low and any modifications boil down to extending the lexica, which makes the development for non-experts straightforward. Further, the first-level grammar does not rely on morphological information and uses approximately 40 fine-grained token types (e.g., word-with-hyphen, word-with-apostrophe, all-capital-letters), which are to a large extent language-independent. To sum up, the majority of the first-level grammar rules used for English is language independent. Clearly, some of them might not be applicable for other languages due to the differences in syntactic structure, but they are intuitively easily modifiable in order to tackle other languages.

The second-level subgrammar consists of patterns for extracting partial information on events: actors, victims, etc. Since the event extraction system is intended to process news articles which refer to crisis-related events, the second-level grammar models only domain-specific language constructions. Moreover, the system processes news clusters which contain articles about the same topic from many sources, which refer to the same event description with different linguistic expressions. This redundancy mitigates the effect of phenomena like anaphora, ellipsis and complex syntax. Consequently, the system processes only the first sentence and the title of each article, where the main facts are summarized in a straightforward manner, usually without using coreference, subordinate sentences and structurally complex phrases. Therefore, the second-level grammar models solely simple syntactic constructions by using 1/2-slot extraction patterns like the ones below. The role assignments are given in brackets.

[A] PER-GRP <DEAD> "were killed"	[F] PER <DEAD> "was shot by" PER <PERPETRATOR>
[B] {PER PER-GRP} <DEAD> "may have perished"	[G] "police nabbed" PER <ARRESTED>
[C] PER-GRP <DEAD> "dead"	[H] PER <KIDNAPPED> "has been taken hostage"
[D] PER <WOUNDED> "was found injured"	[I] PER <RELEASED> "was released"
[E] PER <PERPETRATOR> "has opened fire on"	[J] PER-GROUP <DISPLACED> "fled their homes"

These patterns are similar in spirit to the ones used in AutoSlog [14]. For their creation a blend of machine learning and knowledge-based techniques has been applied. In particular, the learning phase exploits clustered news, which intuitively guarantees good precision. For details see [15].

The fact that in English the word ordering is more strict and the morphology is simpler than in other languages contributes also to the coverage and accuracy of the patterns, which encode non-sophisticated event-description phrases. Consider as an example an article about bank robbery, published by BBC on 16 May 2008 with the following title and first sentence resp.

Eight dead in Philippines (1)

At least eight people have been shot dead during a bank robbery in the Philippines (2)

As for the title (1) the pattern {PERSON | PERSON-GROUP} <DEAD> "dead" will be triggered, where PERSON-GROUP matches *eight*. In the first sentence of the article (2), the pattern PERSON-GROUP <DEAD> "have been shot dead" is triggered, where PERSON-GROUP matches *at least eight people*. The rest of the article proceeds with more details on the victims and the committed crime:

Seven of the victims were employees of the bank in the town of Cabuya, south of Manila, and one was a security guard, police said. Their bodies were discovered when the bank failed to open on time in the morning. Police said two security guards were missing and were suspected of involvement in the robbery... (3)

A proper analysis of this fragment would require to tackle more complex phenomena, e.g., in order to recognize *a security guard* as one of the victims, a correct resolution of ‘one’ in ‘one was a security guard’ is indispensable. Further, a syntactic analysis is needed to capture the subject relation between *two security guards* and the phrase *were suspected of involvement*, which may be used to introduce suspected perpetrators. However, our system discards the text, which goes beyond the first sentence for the following reasons: (a) handling the aforementioned language phenomena is hard and might require knowledge-intensive processing, (b) the crucial information we seek for is included in the title or first sentence, and (c) if some relevant information has not been captured from one article in the cluster we might extract it from other article in the same cluster.

In order to keep the grammar concise and as much as possible language independent, all surface-level linear patterns are represented as pattern types, which indicate the position of the pattern with respect to the slot to be filled (left or right). To be more precise, all patterns are stored in a domain-specific lexicon (applied prior to grammar application), where surface patterns are associated with their type, the event-specific semantic role assigned to the entity which fills the slot (e.g., DEAD, PERPETRATOR) and the number of the phrase which may fill the slot (e.g., plural). For instance, the surface pattern “shot to death” for recognizing dead victims as a result of shooting, is encoded as:

```
shot to death [TYPE: right-context-sg-and-pl, SURFACE: "shot to death", SLOTTYPE: DEAD]
```

The value of the TYPE attribute indicates that the pattern is on the right from its slot (*right-context*), which can be filled by a phrase which refers to one or more humans (*sg-and-pl*). The event-specific semantic role, assigned to each NP filling the slot is DEAD. Through such an encoding of the event-triggering linear pattern, the patterns [A], [B] and [C] from the example above are merged into one:

```
dead-person :-> person-group
gazetteer & [TYPE: right-context-sg-and-pl SLOT: dead] -> ...
```

We have found about 3000 event-triggering surface patterns for English. Due to the strict word order and simple morphology, pattern variants were generated easily.

4. Adapting Grammars to Italian

In the process of adapting our extraction grammar to languages belonging to different language groups, such as Italian, we faced a number of issues.

4.1. Tokenization and Morphological Analysis

At the level of linguistic preprocessing, while some language-specific fine tuning was required, namely on punctuation marks and diacritics, the language independent tokenizer

built for English proved to be to a large extent adaptable to Italian. However, the main shortcomings on its deployment on Italian text was that, in that language, the phonetic phenomenon of elision causes a very productive use of the apostrophe to string together consecutive words (e.g. *dello aviatore* (*of the aviator*) becomes *dell'aviatore*, etc.) resulting in ‘internally apostrophized tokens’ which clearly cannot be listed in the morphological dictionary. We solved this problem keeping the English tokenization process as it is, and then deploying a language-specific component in the morphology, which fires upon failure of simple lexicon look-up. In short, it first tries to split the whole token on the apostrophe occurrence (keeping the apostrophe in the left subtoken) and perform lexicon lookups on the two resulting subtokens. If it fails on either of them, it uses a small set of heuristic rules to guess some basic morphological information of one based on information about the other, like in the sample rules below:

```
[a] Sp U --> Sp N    (e.g. "dell'aviatore")
[b] U N#1#2 --> A#1#2 N#1#2    (e.g. "prim'ordine" ("first order"))
```

Here, [a] states that if an ‘unknown’ subtoken (U) is preceded by a subtoken, which constitutes a ‘shortened’ preposition, then it is labeled as a Noun (N); analogously (in [b]), if an unknown subtoken precedes one which has a noun interpretation, then it is labeled as an adjective (A), with the agreed values (variables #1 and #2) for number and gender, respectively. It is important to note that we use MULTEXT [16] dictionaries for performing morphological analysis, mainly due to the fact that MULTEXT tags are uniform for all languages. We have extended the original MULTEXT dictionary for Italian with circa 7000 entries relevant for the domain of violence and disasters.

4.2. Extraction Patterns

For developing a first release of the Italian event extraction system we followed the original pure surface-level pattern approach. The baseline performance of this version was promising. Nonetheless, performance analysis showed several drawbacks on applying surface patterns on Italian text. First, relative inflectional richness of Romance languages compared to English gives rise to morphological variation along dimensions such as Gender, Number, Person, like in the following set of patterns, capturing a DEAD role, filling the person entity on their left:

```
PERSON <DEAD> e stato ucciso
                [has been killed/was killed, masculine, singular]

PERSON <DEAD> e stata uccisa
                [has been killed/was killed, feminine, singular]

PERSON-GROUP <DEAD> sono stati uccisi
                [has been killed/was killed, masculine, plural]
```

Together with lexical variation, inflection can give rise to a high number of combinations at the level of word forms. This suggests to generalize the extraction patterns using morphological features, rather than relying on the surface level only.

Secondly, Italian verb phrases describing events show some additional structural complexity at the linear level due to the encoding of some essential information on the event, like the ‘means’ or ‘instruments’ of a killing act, in prepositional phrases, as opposed to English which typically uses lexical content of the main verb itself to convey such information. This is clearly shown in these sample excerpts from Italian and En-

glish articles from cross-lingual clusters about the same event returned by the EMM news aggregation system:

excite-news Monday, May 19, 2008 10:43:00 AM CEST (ANSA) - (4)

*MANILA, 19 MAG - Un uomo **ha crivellato a colpi di mitra** alcune case di Calamba, una citta' vicino Manila, uccidendo otto persone....*

*cnn Monday, May 19, 2008 5:58:00 AM CEST A man **strafed** several* (5)

houses during a shooting spree early Monday in a town south of Manila, killing eight people and wounding six others...

Aligned constructions in bold carry approximately the same meaning, but they structure it in a very different way, with the Italian being more verbose. Moreover, Italian more frequently allows adverbial modification in certain positions within verb phrases, and we noticed that this often blocked the triggering of (otherwise recurrent) linear patterns, like in the following text:

tg5 Saturday, February 02, 2007 08:19:00 AM CEST Due donne ital- (6)

*iane **sono state barbaramente uccise a colpi di pietra** sull'isola di Sal, nell'arcipelago di Capo Verde. La drammatica testimonianza della loro amica Agnese, miracolosamente sopravvissuta e ricoverata in stato di choc. (Two Italian women **were barbarically stoned to death on Sal island...**)*

Our prospective solution for these shortcomings follows a strategy consisting of keeping the second-level rules as language independent as possible, dealing with language specific phenomena in the first-level subgrammar. In this view, second-level subgrammar rules are simple, based mainly on positional relations of the extraction patterns with respect to recognized person entities. However, this implies that we should capture productivity and linear complexity within the extraction patterns themselves, and this in turn makes it less feasible to encode extraction patterns as simple domain-specific lexicon entries. Consequently, we started a partial re-engineering of the grammar resources for Italian. Extraction patterns (surface level patterns) were converted from the level of domain-specific lexicon entries into first-level grammar rules; the RHS structures of these rules encode at least the same information as the current surface patterns, and LHSSs rely on both surface and morphological information. In our morphological dictionary we extend the MULTEXT annotation of each domain-specific verb and noun with a semantic attribute SEM, which encodes the semantic role which this verb or noun introduces and the corresponding syntactic position. In such a way, we can generalize on verbs and nouns with similar semantics. To see how all this works, consider how the following set of patterns is converted into the single rule (rule1):

```
PERSON <DEAD> e stato ucciso
[was killed, masculine, singular]

PERSON-GROUP <DEAD> sono rimasti uccisi
[were killed, masculine, plural]

PERSON-GROUP <DEAD> sono state barbaramente uccise
[were barbarically killed, feminine, plural]

PERSON-GROUP <DEAD> sono stati uccisi a pugnate
[were stabbed to death, masculine, plural]

PERSON-GROUP <DEAD> sono stati freddati a colpi di mitra
[were shot to death using a machine gun, masculine, plural]
```

```

rule1 :- (verb & [BASE:"essere", TYPE:"a", VFORM:"i", TENSE:"p", NUMBER:#num1]
  (verb & [BASE:"essere", TYPE:"a", VFORM:"p", TENSE:"s",
    GENDER:#gen1, NUMBER:#num2]
    | verb & [BASE:"rimanere", TYPE:"m", VFORM:"p", TENSE:"s",
      GENDER:#gen1, NUMBER:#num2])
  (adverb)?
  (verb & [VFORM:"p", TENSE:"s", GENDER:#gen2,
    NUMBER:#num3, SEM:"dead-obj"])
  (adverb)?
  ((adposition & [BASE:"a"]
    | (adposition & [BASE:"con"]
      determiner))
    (noun [SEM:"weapon-stroke"]
      | (noun & [BASE:"colpo"]
        adposition & [BASE:"di"]
        noun & [SEM:"weapon"])))? ) : passive-construction
-> passive-construction: pattern & [PTYPE:right-context-violence-event-sg-pl,
  SLOTTYPE:DEAD, NONUM:false,
  NUM:#num1, GEN:#gen1],
  & equal(#num1,#num2,#num3),
  & equal(#gen1,#gen2).

```

Here morphological generalization on the patterns is achieved using FFSs instead of surface forms. As an example, the auxiliary verb group for the passive form in Italian is expressed through constraints on the base form (it must be an *‘essere’* verb), type (*‘a’* for auxiliary), form (*‘i’* for indicative), and tense (*‘p’* for past), followed by a past participle of *‘essere’* or *‘rimanere’* and so on, while some other features like NUMBER are assigned variables for agreement check, etc. Lexical variation is captured by restricting the event-triggering verbs to the class of the ones with *dead-obj* value for their SEM attribute, that is verbs which introduce a DEAD role as the filler of their *object* position. In our simplified linear grammar, this is equivalent to saying that they can appear in *right-context-violence-event* patterns of the passive form, like in the rule above, or in *‘left-context’* patterns of the active form. Below is an example of generalizing *‘left-context’* patterns with the grammar rule *rule2*.

```

uccide {PERSON | PERSON-GROUP} <DEAD>
[kills]

hanno barbaramente ucciso {PERSON | PERSON-GROUP} <DEAD>
[have barbarically killed]

assassinato {PERSON | PERSON-GROUP} <DEAD>
[has assassinated]

rule2 :- ((verb & [BASE:"avere", TYPE:"a", VFORM:"i", TENSE:"p", NUMBER:#num1])?
  (adverb)?
  verb & [VFORM:"p", TENSE:"s", SEM:"dead-obj"]
  (adverb)?
  ((adposition & [BASE:"a"]
    | (adposition & [BASE:"con"]
      determiner))
    (noun [SEM:"weapon-stroke"]
      | (noun & [BASE:"colpo"]
        adposition & [BASE:"di"]
        noun & [SEM:"weapon"])))? ) : active-construction
-> active-construction: pattern & [PTYPE:left-context-violence-event-sg-pl,
  SLOTTYPE:DEAD, NONUM: false, NUMBER:#num1].

```

An analogous generalisation might be achieved for nominal patterns, annotating event-related nouns for the ROLE they introduce at the right of their pattern as SEM:*dead-obj*. Here is a sample list of such patterns followed by a rule (*rule3*), which generalizes them:

```

morte di {PERSON | PERSON-GROUP} <DEAD>
[death of]

morte improvvisa dei PERSON-GROUP <DEAD>
[sudden death of the, masculine, plural]

```



```

morte delle PERSON-GROUP <DEAD>
[death of the, feminine, plural]

omicidio della {PERSON | PERSON-GROUP} <DEAD>
[homicide of the, feminine, plural]

strage di PERSON-GROUP <DEAD>
[slaughter of]

rule3 :-> ((noun & [SEM:"dead-obj*"])
(adjective)?
adposition & [BASE: di, GENDER:#gen, NUMBER:#num]) : nominal-pattern
-> nominal-pattern: pattern & [PTYPE:left-context-violence-event-sg-pl,
SLOTTYPE:DEAD, NONUM: FALSE,
NUMBER:#num, GEN:#gen].

```

Achieving a higher degree of generalization by using more linguistically-sophisticated rules comes at the cost of introducing some complications at the level of more language specific linguistic resources. Nonetheless, we emphasize two points. First, several of the generalizations on extraction patterns are language independent, particularly within the same language family. For example, the last rule shown above is portable to French, while it runs only slightly different for English. That means, provided that an analogous annotation of morphological resources is available, the rule approach reduces human work in the process of extension to new languages. Most importantly, the described approach to generalization is meant to capture linguistic regularities in expressing domain-specific semantic roles, rather than capturing generic syntactic structure. For instance, the optional subpattern at the end of *rule1* and *rule2* is combined with a suitable dictionary annotation for domain-specific semantic categories in order to capture how an ‘instrument’ or ‘means’ role of a violent act can be expressed in a class of varying prepositional phrases. Namely, SEM: "weapon" is used to label nouns referring to weapons, while SEM: "weapon-stroke" is a rather language-specific class of morphologically derived nouns referring to stroke produced with a weapon (*‘pugnale’* (stab) becomes *‘pugnalata’* in Italian).

The application of the subgrammar for extracting the event-triggering phrases (e.g., the rules *rule1*, *rule2* and *rule3*), is applied in parallel with the named entity grammar since no overlapping should occur on text between the two classes of patterns.

Notice also that in some of the rules (e.g., in *rule1*) we do perform an agreement check (a call to *equal FO*). Agreement information might help disambiguate some ambiguous constructions in relatively inflection-rich languages like Italian. For instance, without an agreement in the following sentence the positional relation between the right pattern *assassinato* and the complex person group phrase *tre guardie del corpo di J.F. Kennedy* would trigger a second-level rule, which erroneously extracts an event description about the killing of J.F.K.’s bodyguards.

tre guardie del corpo di J.F. Kennedy, assassinato a Dallas... (three (7)
bodyguards of J.F. Kennedy, assassinated in Dallas...)

However, in this case the longest match strategy on person group extraction is blocked because of NUMBER information incompatibility with the extraction pattern, so the system can discard the uncorrect DEAD role extraction and back off to the person name *J.F. Kennedy* (the correct interpretation). Agreement check is virtually attainable also within the surface approach by suitable, manual annotation of lexical resources. In a morphology-based approach, though, agreement-related information is much more nat-

Table 1. Comparison of the baseline and hybrid system

	<i>Dead</i>		<i>Injured</i>		<i>Actor</i>		<i>Arrested</i>	
	BL	LR	BL	LR	BL	LR	BL	LR
P	0.82	0.91	0.66	0.77	0	0.66	0.83	0.70
R	0.48	0.84	0.15	0.53	0	0.25	0.55	0.66
F ₁	0.60	0.87	0.24	0.62	0	0.36	0.66	0.67

urally passed up to output structures from the phrase's head component (e.g. from the main verb in *rule2*). This allows rules for person entity and extraction pattern recognition to keep naturally enforcing agreement constraints while covering more and more structurally complex constructions.

We carried out a first evaluation of the performance improvement resulting from the grammar restructuring required by adapting of surface level pattern approach to Italian.

5. Evaluation

In order to evaluate the coverage and accuracy of the Italian event extraction system, we gathered test data by downloading EMM article clusters during 4 consecutive days in July, 2008. The final evaluation corpus contained 213 clusters, only a part of which were about violent event stories. While this procedure guarantees real world data, it suffers from data sparseness problems, as some of the roles may not be instantiated in text, due to the relatively small corpus size. That was the case in our experiment for *KIDNAPPED* and *RELEASED* roles, which therefore we did not report in the final evaluation. On this corpus, we ran both the baseline version of the system, namely the one based on linear patterns, and a hybrid version containing the more abstract rules together with a subset of highly irregular linear patterns, which were not converted into more abstract rules. The rationale for this is that we deploy abstract rules in order to deal with productive, structurally complex language constructions, while backing off to surface level patterns to cover more idiomatic constructs. We denote the baseline and hybrid system with BL and LR respectively. Table 1 shows a comparative evaluation of the two Italian event extraction systems. In particular, we measured Precision (P), Recall (R) and F-measure for each role.

Evaluation was done separately for each role, and the data was collected cluster by cluster. Namely, for each cluster of articles we record whether it contains a reference to the filler of a specific role; then we record whether the system detected any filler whatsoever for that role, and finally, we record a correct detection if the role filler description returned equals at least one of the descriptions occurring in the cluster.

The main performance gain we could observe was clearly in recall. A thorough analysis revealed that it was mainly due to morphological abstraction and the productivity achieved through more abstract patterns. In particular, one case was interesting. *ACTOR* role fillers are usually the hardest to be extracted from Italian text, as their relationship with the main event verb is either to be derived through deep semantic or even pragmatic inferences, or extracted through complex two-slot patterns. The following two text samples from the evaluation corpus illustrate examples of extracted and non-extracted *ACTOR* role instances.

35enne ucciso a coltellate da tre uomini nel ravennate (8)
(35 years old stabbed to death by three men in Ravenna area ...)

Andrea Tartari, 35enne bolognese, e stato accoltellato a Ravenna durante un diverbio con tre giovani appoggiati alla sua auto (9)
(Andrea Tartari, 35 years old from Bologna, was stabbed in Ravenna during a quarrel with three young men leaning...)

In the latter case, extracting *tre giovani* (the three young men) as ACTOR of the stabbing event would depend on inferring the stabbing itself as being the outcome of the ‘quarreling’ event, which is hard to perform for a shallow information extraction system. In the former case the hybrid system managed to capture the ACTOR by triggering a two-slot extraction pattern, while the baseline system failed. As can be seen from example 8, the Italian language tends to place the information about the means of a killing event between the verb and the ACTOR; this is done in such a productive way, that it cannot be expressed in a list of surface-level patterns. On the other hand, structure-based rules, together with a proper semantic annotation of the lexicon succeed to recognize such constructions.

Precision gain was smaller. In one particular case a deterioration in precision could be observed, namely in the case of the ARRESTED role. Analysing the cases which contributed to the deterioration of the precision showed the possible risk on the shift towards an approach based on morphological abstraction. In the following text snippet from the corpus, the Italian word form *giustizia* has a noun reading (justice), and a verb reading (to execute, 3rd person present indicative). This word is a member of the verb class allowed in patterns for extracting the DEAD role. Consequently, the system wrongly extracted the proper name *Angelino Alfano* as a victim.

Il ministro della Giustizia Angelino Alfano ha commemorato stamane a Palermo... (10)
(The minister of Justice Angelino Alfano commemorated this morning in Palermo...)

Given that our system does not perform any word sense disambiguation, it cannot get rid of contextually irrelevant morphological descriptions sometimes produced by the grammar application. Clearly, such ambiguities could be resolved by applying some filtering mechanism (e.g., additional lexicon annotation and/or using some heuristics), which is not currently being deployed.

Finally, in a number of cases, numerals which are part of temporal expressions are mistakenly captured as quantity expressions in active patterns, e.g., in:

un commando terrorista ha rapito due giorni fa un giornalista italiano... (11)
(a terrorist commando kidnapped two days ago an Italian journalist...)

the system extracted *due* (two) as kidnapping victim description. Temporal expressions are quite pervasive in event descriptions as they encode a crucial semantic dimension, and in Italian they tend to be more freely located in text. In order to alleviate the aforementioned problem, we currently apply some structural patterns to detect unambiguous temporal expressions, which helps in resolving ambiguities like the aforementioned one. Structural complexity and productivity of temporal expressions makes it hard to list them as surface level patterns.

We also compared the performance of the English event extraction system with the Italian system based on linguistic rules on two news corpora in English and Italian downloaded at the same time from EMM. There was some overlapping in the topics in the two languages, however the news were quite different in general. Therefore, an accurate comparison between the English and Italian systems was not possible. However, it is worthwhile to mention that for the role *DEAD* the English system achieved F_1 of 0.9 and the Italian achieved nearly the same result, namely 0.87.

6. Conclusions

We reported on the process of adapting an existing real-time event extraction engine for English based on a cascade of finite-state grammars to processing Italian news articles.⁶ We utilize an IE-oriented efficient finite-state formalism, which was sufficient to encode extraction grammars for detecting events from real-world news since hard-to-tackle linguistic phenomena can be discarded by approaching event extraction in a cluster-centric manner. Moreover, the pattern formalism turned out to be more amenable for non-linguist grammar writers than other ones we tested. Although from the scientific point of view the work presented in this paper might not be considered as very novel, we strongly believe that some of our somewhat technical-in-nature observations and solutions might constitute interesting guidelines and hints for solving and coping with similar tasks in a multilingual text processing environment, which processes a vast amount of textual data in real time.

The English event extraction system presented here relies mainly on a multilingual named entity recognition grammar and a language-specific dictionary of surface-level event extraction patterns, whose encoding does not require expert linguistic knowledge. However, more abstract grammar rules are necessary for a better coverage of Italian, because of the free word order and morphological richness of this language.

Consequently, we designed a slightly more linguistically sophisticated Italian extraction grammar, which covers a rich variety of morphological and syntactic constructions. Nevertheless, we try to keep the linguistic descriptions as simple as possible and at the same time reduce the effort for engineers maintaining the system. In particular, we achieve this by: (i) providing a higher, grammar layer, which is weakly language-dependent, and (ii) improving the maintainability by separating the domain-lexica from the grammar rules, where the domain-specific lexica constitute the only resource, which is the subject of potential modification by the system users (e.g., adapting to a new domain, enriching the coverage of the lexica).

Our experiment demonstrated that the linguistically more sophisticated grammar improves significantly the performance of the event extraction system for Italian. We believe that the Italian grammar can be adapted with its current structure to other Romance languages straightforwardly. The live event extraction system for Italian is publicly accessible at: <http://press.jrc.it/geo?type=event&format=html&language=it>. For the English version change the value of the language attribute to *en*.

⁶We would like to thank Bruno Pouliquen for providing invaluable resources for Italian. Further, we are also indebted to Ralf Steinberger and Jonathan Brett Crawley for supporting our work with many comments and suggestions. The presented work could not have been possible without the work done by Martin Atkinson, Erik van der Goot and many other EMM colleagues.

References

- [1] N. Ashish, D. Appelt, D. Freitag, and D. Zelenko. *Proceedings of the Workshop on Event Extraction and Synthesis, held in conjunction with the AAAI 2006 conference, Menlo Park, California, USA*. 2006.
- [2] R. Grishman, S. Huttunen, and R. Yangarber. Real-time Event Extraction for Infectious Disease Outbreaks. In *Proceedings of HLT 2002*, San Diego, USA, 2002.
- [3] G. King and W. Lowe. An Automated Information Extraction Tool For International Conflict Data with Performance as Good as Human Coders: A Rare Events Evaluation Design. *International Organization*, 57:617–642, 2003.
- [4] C. Best, E. van der Goot, K. Blackler, T. Garcia, and D. Horby. Europe Media Monitor. Technical Report EUR 22173 EN, European Commission., 2005.
- [5] C. Aone and M. Santacruz. REES: A Large-Scale Relation and Event Extraction System. In *Proceedings of ANLP 2000*, 2000.
- [6] H. Tanev, J. Piskorski, and M. Atkinson. Real-Time News Event Extraction for Global Crisis Monitoring. In E. Kapetanios, V. Sugumaran, and M. Spilipoulou, editors, *Proceedings of NLDB 2008*, pages 207–218. LNCS 5039, Springer, 2008.
- [7] J. Piskorski, H. Tanev, M. Atkinson, and E. Van der Goot. Cluster-Centric Approach to News Event Extraction. In *Proceedings of the 6th International Conference on Multimedia & Network Information Systems*, Wroclaw, Poland, 2008.
- [8] J. Piskorski. CORLEONE – Core Linguistic Entity Online Extraction. Technical report 23393 EN, Joint Research Center of the European Commission, Ispra, Italy, 2008.
- [9] J. Piskorski. ExPRESS – Extraction Pattern Recognition Engine and Specification Suite. In *Proceedings of the International Workshop Finite-State Methods and Natural language Processing 2007 (FSMNLP’2007)*, Potsdam, Germany, 2007.
- [10] W. Drożdżyński, Krieger. H.-U., J. Piskorski, U. Schäfer, and F. Xu. Shallow Processing with Unification and Typed Feature Structures — Foundations and Applications. *Künstliche Intelligenz*, 2004(1):17–23, 2004.
- [11] H. Cunningham, D. Maynard, and V. Tablan. JAPE: a Java Annotation Patterns Engine (Second Edition). Technical Report, CS–00–10, University of Sheffield, Department of Computer Science, 2000.
- [12] B. Popov, A. Kiryakov, D. Ognyanoff, D. Manov, A. Kirilov, and M. Goranov. Towards Semantic Web Information Extraction. In *Proceedings of ISWC, Sundial Resort, Florida, USA*, 2003.
- [13] R. Yangarber and R. Grishman. Machine Learning of Extraction Patterns from Un-annotated Corpora. In *Proceedings of the 14th European Conference on Artificial Intelligence: Workshop on Machine Learning for Information Extraction, Berlin, Germany*, 2000.
- [14] E. Riloff. Automatically Constructing a Dictionary for Information Extraction Tasks. In *Proceedings of the 11th National Conference on Artificial Intelligence*, 1993.
- [15] H. Tanev and P. Oezden-Wennerberg. Learning to Populate an Ontology of Violent Events. In Fogelman-Soulie, F. and Perrotta, D. and Piskorski, J. and Steinberger, R., editor, *Mining Massive Data Sets for Security*. IOS Press, 2008.
- [16] T. Erjavec. MULTEXT - East Morphosyntactic Specifications. URL: <http://nl.ijs.si/ME/V3/msd/html/>, 2004.

This page intentionally left blank

Short Papers

This page intentionally left blank

Finite State Models for the Generation of Large Corpora of Natural Language Texts

Domenico CANTONE, Salvatore CRISTOFARO, Simone FARO and
Emanuele GIAQUINTA

Università di Catania, Dipartimento di Matematica e Informatica
Viale Andrea Doria 6, I-95125 Catania, Italy
{cantone | cristofaro | faro | giaquinta}@dmi.unict.it

Abstract. Natural languages are probably one of the most common type of input for text processing algorithms. Therefore, it is often desirable to have a large training/testing set of input of this kind, especially when dealing with algorithms tuned for natural language texts. In many cases the problem due to the lack of big corpus of natural language texts can be solved by simply concatenating a set of collected texts, even with heterogeneous contexts and by different authors.

In this note we present a preliminary study on a finite state model for text generation which maintains statistical and structural characteristics of natural language texts, i.e., Zipf's law and inverse-rank power law, thus providing a very good approximation for testing purposes.

Keywords. finite state model, automaton, natural language generation, language identification, text processing.

Introduction

Natural languages are probably one of the most common type of input for text processing algorithms. Therefore, it is often desirable to have a large training/testing set of input of this kind, especially when dealing with algorithms tuned for natural language texts. The problem in creating good corpora is that often natural language texts are too short with respect to the dimension required to test effectively the goodness of text processing algorithms, such as string matching and compression algorithms. This is, for instance, the case of the well-known Canterbury Corpus [1], used for testing lossless data compression algorithms, which contains natural language texts with a relative small dimension of not more than 500Kb. The only exception is the "King James Version of the Bible" (approximately 3, 85Mb) contained in the Large Corpus [1]. On the other hand corpora of non-textual data contain test files with dimensions up to 3Mb (like the Protein Corpus [2] and the Silesia Corpus [3]), while testing on random texts is often performed on buffers of dimension 10Mb [4] and 20Mb [5].

In many cases the problem due to the lack of big corpus of natural language texts can be solved by simply concatenating a set of collected texts, even with heterogeneous contexts and by different authors. This is the case, for example, of The *Linguistic Data Consortium* (<http://www ldc.upenn.edu>), an open consortium of universities which creates, collects and distributes speech and text databases and other resources for research and development purposes.

However, in this context, the task of being able to automatically generate texts which maintain properties of real texts is appealing. In this note we present a preliminary study on a finite state model for text generation which maintains statistical and structural characteristics of natural language texts, i.e., Zipf's law [6] and inverse-rank power law [7], thus providing a very good approximation for testing purposes.

1. Preliminaries

Before entering into details, we review a bit of notations and terminology. A string S of length $m > 0$ is represented as a finite array $S[0..m-1]$. The length of S is denoted with $|S|$, i.e., $|S| = m$. By $S[i]$ we denote the $(i+1)$ -st character of S , for $0 \leq i < m$. Likewise, by $S[i..j]$ we denote the substring of S contained between the $(i+1)$ -st and the $(j+1)$ -st characters of S , for $0 \leq i \leq j < m$.

A FINITE STATE AUTOMATON is a quintuple $\mathcal{A} = (Q, p_0, F, \Sigma, \delta)$, where Q is the set of states of the automaton, $p_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of accepting states, Σ is the alphabet of characters labeling transitions, and δ is a partial function from $Q \times \Sigma$ to Q , called the transition function. If $\delta(p, c)$ is not defined for a state $p \in Q$ and a character $c \in \Sigma$, we say that $\delta(p, c)$ is an empty transition and write $\delta(p, c) = \perp$. Moreover, for all $c \in \Sigma$ we put $\delta(\perp, c) = \perp$.

In contrast with what can be observed in random texts with a uniform character distribution, it turns out that some naturally occurring phenomena in natural language texts obey a *power-law* distribution.

Zipf's law (cf.[6]), named after the Harvard linguistic professor George Kingsley Zipf (1902-1950), is one of the most interesting applications of the power-law to natural languages. In particular, Zipf's law connects the rank of a word in a natural language text with its relative frequency on the text itself as follows: given a text T , Zipf's law states that, with a very good approximation, the relative frequency of a word is inversely proportional to its rank. More formally, if R is the number of different words in T , then the relative frequency $f(r)$ of a word with rank r in T is approximated by expression (1) shown below:

$$(1) \quad f(r) \simeq \frac{1}{r \ln(1.78R)}, \quad (2) \quad f(r) \simeq \frac{(R-i+1)^k}{\sum_{j=1}^R j^k}.$$

Figure 1 presents the relative frequencies of words in a random text buffer, with a uniform distribution of characters (Figure 1.A) and in a natural language text "Hamlet" (Figure 1.B) and their approximations with the Zipf's law. In contrast with the natural language text, we can observe that the relative frequency of words in the random text does not follow a Zipf's law.

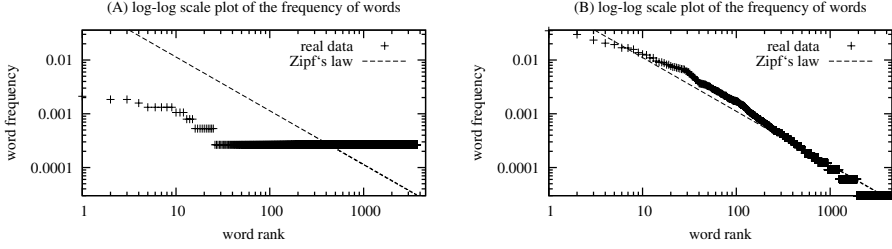


Figure 1. The relative frequencies of words in a random text buffer (A) and in a natural language text (B), and their approximations with Zipf's law. The text buffer in (A) has been generated randomly with a uniform distribution of characters ($n = 200000, \sigma = 50$). The natural language text in (B) derives from the English drama "Hamlet" ($n = 174073, \sigma = 65$).

Recently, in [7] a similar characterization has been reported for the frequencies of characters in natural language texts. Such distribution model gives a very good approximation of the relative frequency function of characters in terms of their rank both in natural language dictionaries and texts. The model is based on the following **inverse-rank power-law** of degree k , which states that if R is the number of different characters in T , then the relative frequency $f(r)$ of the character with rank r in T is approximated by the above expression (2), for a degree $k \in \mathbb{R}$ whose value is to be determined experimentally (usually k ranges in the closed interval $[3..10]$).

2. The Finite State Model

The model adopted in this note is a Deterministic Probabilistic Finite Automaton (DPFA) [8,9], called *Extended q -Gram Model* [2], which inherits the statistical structure of the string used for its construction (see below). The q -Grams automata are equivalent to a class of DPFA known as *stochastic k -testable automata* [10].

The DPFA's are models which are generative in nature. This is in contrast with the standard definition of automata in the conventional (non-probabilistic) formal language theory, where strings are generated by grammars, whereas the automata are the accepting devices. Thus, if S is a natural language string, then a DPFA for S can be used to generate random texts which maintain the peculiar characteristics of S itself. The q -Gram Model for a string S is constructed by extracting all q -grams of the string S , for some fixed $q > 0$, and by carrying the statistical relationships between overlapping q -grams.

To begin with, let S be a string and let Σ be its alphabet. Given a positive integer q , with $q \leq |S|$, we define a q -gram of S as a substring w of S of length q , i.e., $w = S[i..i+q-1]$, for some $0 \leq i \leq |S|-q$. We denote with $G_q(S)$ the set of all q -grams of S . We define also an occurrence function ρ_S which associates to each nonempty string w over Σ the number of its occurrences in S , namely:

$$\rho_S(w) = |\{i : 0 \leq i \leq |S| - |w| \text{ and } S[i..i+|w|-1] = w\}|.^1$$

¹Notice that $0 < \rho_S(w) \leq |S| - q + 1$, for each q -gram w of S .

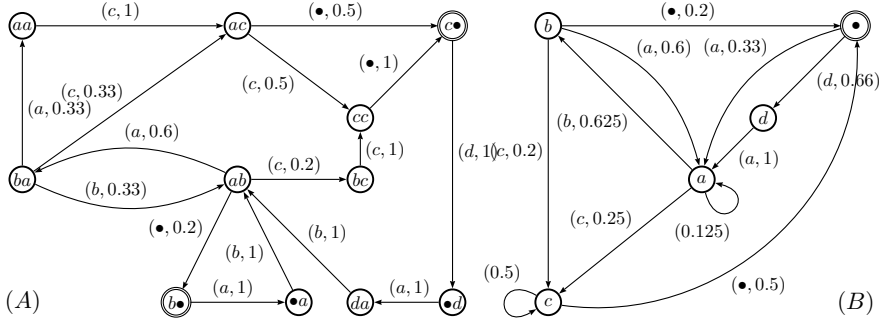


Figure 2. (A) The 2-GA for the string $S = abaac \bullet dabab \bullet abacc \bullet dabcc$, where $\Sigma = \{a, b, c, d, \bullet\}$ and $\Sigma_{sep} = \{\bullet\}$. Each transition $\delta(w, c)$ of the automaton is labeled with the pair $(c, \varphi(w, c))$, where c is the character which performs the transition and $\varphi(w, c)$ is the relative frequency of the transition. (B) The 1-GA for the string $S = abaac \bullet dabab \bullet abacc \bullet dabcc$.

A standard natural language text can be seen as a sequence of words separated by special symbols such as punctuation marks, numbers, *blanks*, etc. Thus, we assume that there is a distinguished set $\Sigma_{sep} \subseteq \Sigma$ containing such symbols, used as separators between different words.

Given a value q , with $0 < q < |S|$, the q -Gram Automaton (q -GA for short) for the string S is formally defined as the probabilistic finite state automaton $\mathcal{A} = (Q, p_0, F, \Sigma, \delta, \varphi)$, where

1. Q is the set of all q -grams of S , i.e., $Q = G_q(S)$;
2. p_0 is the initial state, defined as the first q -gram of S , i.e. $p_0 = S[0..q-1]$;
3. F is the set of final states, defined as $F = \{w \in Q \mid w[q-1] \in \Sigma_{sep}\}$;
4. δ is the transition function defined, for each $w \in G_q(S)$ and $a \in \Sigma$, by

$$\delta(w, a) = \begin{cases} w[1..q-1].a & \text{if } w[1..q-1].a \in G_q(S) \\ \perp & \text{otherwise,} \end{cases}$$

5. $\varphi : (Q \times \Sigma) \rightarrow \mathbb{R}$ is a map which associates to each transition of the automaton its relative frequency. The map φ is formally defined by

$$\varphi(w, a) = \begin{cases} \frac{\rho_S(w.a)}{\sum_{c \in \Sigma} \rho_S(w.c)} & \text{if } \delta(w, a) \neq \perp \\ 0 & \text{otherwise,} \end{cases}$$

for each $w \in G_q(S)$ and $a \in \Sigma$.

See Figure 2 for a pictorial illustration. The construction of the q -GA for a string S of length n takes $\mathcal{O}(n + |\Sigma|^{q+1})$ -time and requires $\mathcal{O}(|\Sigma|^{q+1})$ -space, where Σ is the alphabet of the string S .

3. Text Generation

In this section we present a simple algorithm for generating random texts by means of the finite state model described above. Then we present experimental evidence that the random texts so generated obey Zipf's law and the inverse-rank power law, namely they enjoy the structural and statistical characteristics of natural language texts.

<pre> RANDOM-TRANSITION(\mathcal{A}, Σ, w) 1. $r \leftarrow \text{random}(0, 1]$ 2. $\pi \leftarrow 0$ 3. $j \leftarrow 1$ 4. while $\pi < r$ do 5. $\pi \leftarrow \pi + \varphi(w, c_j)$ 6. $j \leftarrow j + 1$ 7. return c_j </pre>	<pre> GENERATOR($S, \mathcal{A}, q, \Sigma, n$) 1. $w \leftarrow S[0..q-1]$ 2. $T[0..q-1] \leftarrow w$ 3. $i \leftarrow q$ 4. while $i < n$ do 5. $c \leftarrow \text{RANDOM-TRANSITION}(\mathcal{A}, \Sigma, w)$ 6. $T[i] \leftarrow c$ 7. $w \leftarrow \delta(w, c)$ 8. $i \leftarrow i + 1$ 9. return T </pre>
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

The algorithm GENERATOR given above takes as input a string S , the q -GA \mathcal{A} for the string S , a dimension q , the alphabet Σ , and the length n of the output text buffer T . We assume that the alphabet Σ is an ordered finite alphabet, so that we can write $\Sigma = \{c_1, c_2, \dots, c_\sigma\}$, with $|\Sigma| = \sigma$. The algorithm starts its transitions from the initial state $w = S[0..q-1]$. Thus the first q characters of the output text T will be equal to the first q characters of S . Then it performs a loop until n characters have been inserted in T . More precisely, at each iteration, it uses the last q -gram of T , w , to compute the subsequent character c to be inserted. In particular c is selected randomly among all possible transitions $\delta(w, c)$ in the q -GA, according to their frequencies $\varphi(w, c)$ (procedure RANDOM-TRANSITION). Then w is updated to $\delta(w, c)$. Clearly the GENERATOR algorithm takes $\mathcal{O}(n)$ -time for computing a text buffer of length n .

The following table lists some files containing random texts (output files), all of dimension 2Mb, generated by the algorithm described above from natural language texts (source files) of different sizes and languages, with grams dimension $q = 2$. Each file is accessible via a URL of the form

http://www.ippari.unict.it/faro/fsmnlp08/file_name.txt.

text	language	source file	output file
Hamlet (W. Shakespeare)	English	ham.txt (176Kb)	hamg3.txt
La Divina Commedia (D. Aligheri)	Italian	div.txt (549Kb)	divg3.txt
De la Terre à la Lune (J. Verne)	French	terlun.txt (335Kb)	terlung3.txt
Don Quijote (M. Cervantes)	Spanish	quijote.txt (2, 04Mb)	quijoteg3.txt
English dictionary (151.160 entries)	English	endict.txt (1, 47Mb)	endictg3.txt
Words beginning with w (328 words)	English	wwords.txt (2, 24Kb)	wwordsg3.txt
Italian dictionary (277.313 entries)	Italian	itdict.txt (3, 13Mb)	itdictg3.txt
World Fact Book (Canterbury Corpus)	English	world192.txt (2, 35Mb)	world192g3.txt
The Bible (Canterbury Corpus)	English	bible.txt (3, 85Mb)	bibleg3.txt

Below are presented three examples of random texts, of length $n \geq 100$, generated by the algorithm described above from strings S_1 , S_2 , and S_3 with grams dimensions $q \in \{1, 2\}$.

Example 1. $S_1 = \text{"abaac dabab abacc dabcc"}$ (this is the string used in Figure 2):

(q=1) *ab daababab ab ababcc abacc dabab dababc dababacccccc abababababaacc dabababab
dababcc ab dab dababa*
 (q=2) *ababcc dab ab abacc dabcc dabac dabaac dabcc dab abaacc dabac dabac dabab abcc dab
abacc dabacc dabc*

Example 2. S_2 = concatenation of the 328 different words of length 6 of the English dictionary, beginning with the letter *w*:

(q=1) *we warif wolviter w wices whofeshs wadeddd wommmmpier wilads waxeathaveshs wally war
wind wis waldooup*
 (q=2) *waffy wra winne wifer wiper whoolver woo wafed weekly wagong whing wincern weaker wrer
woolver woren*

Example 3. S_3 = concatenation of the 1389 different words of length 6 of the Italian dictionary, beginning with the letter *a*

(q=1) *aro acca agnfi ac ara ara alito andereatealbiannna arga aloniacci affi arma ale aluca atsiti affe*
 (q=2) *abbaggia azoiolo apone amperemia abdulsa assi annomie arreso agrai amarpio aucideraspio
aliata apta*

Observe that texts generated by the 1-GAs contain words which are not closely related with the structure of the source string. Short words like “*ab*” in Example 1, “*w*” in Example 2, and “*ac*” in Example 3, appear in the generated texts together with quite long words like “*abababababaacc*” (in Example 1), “*waxeathaveshs*” (in Example 2) and “*andereatealbiannna*” (in Example 3). Moreover, strings containing long sequences of a same character, like “*dababacccccc*” (in Example 1) and “*wommmmpier*” (in Example 2) can occur. Instead, the length of the words generated by the 2-GAs are very close to the length of the words in the source string and anomalies due to character repetitions are not present.

Figure 3 shows the relative frequencies of characters and words in two different text buffers of dimension 2Mb, generated from 2-GAs for two different natural language texts. Observe that for both text buffers the relative frequencies of characters are well approximated by an inverse-rank power law, of degree 4.7 and 5.9, while the relative frequency of words follows closely a Zipf’s law.

4. Conclusion and Plans for Future Works

In this note we have presented a preliminary study of a finite state model for generating random text buffers with the same structure of natural language texts. The model can be used to generate large corpora of data for testing text processing algorithms for data-compression and pattern-matching. We intend to investigate further applications of the Extended q -Gram Model in automatic music generation, for creating original music pieces from input scores, and in the field of image preprocessing for automatic texture generation.

Another major application field which we intend to investigate relates to the *Language Identification* problem, which has applications in many areas such as in spelling and grammar correction, in database search engine, etc. For instance, given a q -GA $\mathcal{A} = (Q, p_0, F, \Sigma, \delta, \varphi)$ and an input string w of length m , we can associate to w a similarity coefficient value, $\Gamma_{\mathcal{A}}(w)$, computed as the average relative frequency of transitions

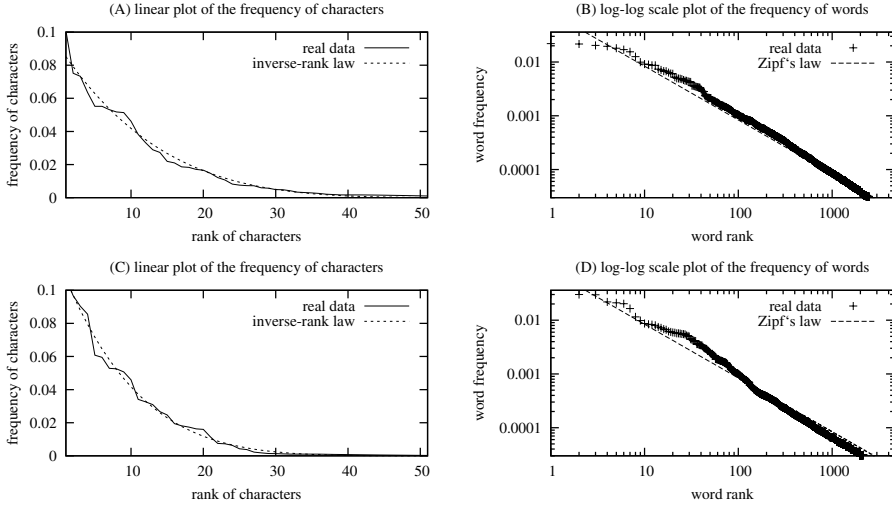


Figure 3. The relative frequencies of characters (A-C) and words (B-D) in two different randomly text buffers of dimension $2MB$, generated from two 2-GA, and their approximations with the inverse-rank power law and Zipf's law, respectively. The text buffer in (A-B) comes from the English drama "Hamlet" ($n = 174073, \sigma = 65$). The text buffer in (C-D) comes from the Italian poem "La Divina Commedia" ($n = 548159, \sigma = 62$).

$(w[i..i+q-1], w[q])$, for $0 \leq i < |w| - q$. A similar application, based on Markov Models, has been presented in [11].

Then the language identification problem can be addressed by parsing the input string with different q -GAs constructed over different natural language texts and taking the language which leads to the higher similarity coefficient value. Figure 4(A-B) presents the similarity coefficient values obtained by parsing a set of English and Italian words with 2-GAs constructed over three different natural language dictionaries whereas Figure 4(C) presents the similarity coefficient values for a set of English, Italian and French proverbs.

References

- [1] A. Ross and T. Bell. *The Canterbury Corpus*. University of Canterbury, New Zeland, 1997. <http://corpus.canterbury.ac.nz>.
- [2] C. G. Nevill-Manning and I. H. Witten. Protein is incompressible. In *Data Compression Conference*, pages 257–266, 1999. <http://data-compression.info/Corpora/ProteinCorpus>.
- [3] S. Deorowicz. *Silesia Corpus*. Silesian University of Technology, Poland, 2003. <http://data-compression.info/Corpora/SilesiaCorpus>.
- [4] C. Allauzen, M. Crochemore, and M. Ranot. Factor oracle: a new structure for pattern matching. In *Theory and Practice of Informatics*, volume 1725 of *LNCS*, pages 291–306, Milovy, Czech Republic, 1999.
- [5] D. Cantone and S. Faro. Fast-search algorithms: New efficient variants of the boyer-moore pattern matching algorithm. In *Experimental and Efficient Algorithms: Second International Workshop, WEA 2003, Ascona, Switzerland, May 26-28, 2003. Proceedings*, volume 2647 of *Lecture Notes in Computer Science*, pages 622–. Springer Berlin, 2003.

(A) words	Italian	English	German	(B) words	Italian	English	German
<i>airships</i>	0.0279	0.1323	0.0576	<i>quadrato</i>	0.3313	0.2491	0.2223
<i>beautiful</i>	0.0989	0.1355	0.1022	<i>quaglia</i>	0.4088	0.2565	—
<i>cetrioli</i>	0.1391	0.1000	0.0610	<i>quando</i>	0.4598	0.2687	0.2894
<i>crazed</i>	0.0362	0.2346	0.0378	<i>quantizzammo</i>	0.4086	0.1884	0.1915
<i>cucumbers</i>	0.0821	0.1871	—	<i>raggi</i>	0.2519	0.1027	0.0639
<i>dirigibili</i>	0.2186	0.1360	0.0949	<i>spokes</i>	0.0938	0.2067	0.1360
<i>equazioni</i>	0.4025	—	—	<i>spring</i>	0.0851	0.2559	0.1900
<i>hydrophily</i>	—	0.3123	—	<i>stare</i>	0.1795	0.0903	0.1377
<i>idrofilo</i>	0.1550	0.0963	0.0536	<i>stars</i>	0.1002	0.2112	0.1209
<i>impazzivo</i>	0.1947	—	0.0691	<i>state</i>	0.2354	0.1738	0.1564
<i>inno</i>	0.4825	0.1958	0.1793	<i>testamenti</i>	0.2631	0.1519	0.1451
<i>meravigliosi</i>	0.2221	0.0965	0.1035	<i>trentesimo</i>	0.2327	0.1168	0.1483
<i>none</i>	0.2018	0.2063	0.1039	<i>why</i>	—	0.1159	—
<i>piuolo</i>	0.1296	—	—	<i>wills</i>	0.0994	0.2656	0.1713

(C) sentences	italian	english	french
<i>A buon cavallo non manca sella</i>	0.2189	—	—
<i>A buon intenditor poche parole</i>	0.2275	—	—
<i>A picture is worth a thousand words</i>	—	0.2132	—
<i>A tavola non si invecchia</i>	0.2630	—	—
<i>A word to the wise is sufficient</i>	—	0.2409	—
<i>Action speak louder than words</i>	—	0.2276	—
<i>Buon sangue non mente</i>	0.2492	—	—
<i>Chi domanda cio che non dovrebbe, ode cio che non vorrebbe</i>	0.2755	—	—
<i>Chi dorme d'agosto, dorme a suo costo</i>	0.1837	—	—
<i>Hard words break no words</i>	—	0.1870	—
<i>Il faut tourner sa langue sept fois dans sa bouche avant de parle</i>	—	—	0.2604
<i>In the land of the blind, the one eyed man is king</i>	—	0.3012	—
<i>La parole est d'argent, mais le silence est d'or</i>	—	—	0.2185
<i>Le parole sono femmine e i fatti sono maschi</i>	0.1822	—	—
<i>Nel mezzo del cammin di nostra vita</i>	0.2672	—	—
<i>The cat will mew and dog will have its day</i>	—	0.2882	—
<i>To be or not to be, that is the problem</i>	—	0.3282	—

Figure 4. (A-B) The similarity coefficient values for a set of words over the Italian, English and German dictionaries. The symbol “—” indicates that the word is not recognized by the automaton. **(C)** The similarity coefficient values for a set of English, Italian and French proverbs. The sentences have been tested with three 2-GAs constructed over three different natural language texts: the English drama “Hamlet” by William Shakespeare; the Italian poem “La Divina Commedia” by Dante Alighieri; the French novel “De la Terre à la Lune” by Jules Verne.

- [6] G. K. Zipf. *Selective Studies and the Principle of Relative Frequency in Language*, volume 23. Harvard University Press, Cambridge, MA, 1932.
- [7] D. Cantone and S. Faro. On the frequency of characters in natural language texts. In *Proc. of the 3rd AMAST Workshop on Language Processing, AMILP 2003*, pages 10–24, 2003.
- [8] A. Paz. *Introduction to Probabilistic Automata*. Academic Press, New York, NY, 1971.
- [9] E. Vidal, F. Thollard, C. De La Higuera, F. Casacuberta, and R.C. Carrasco. Probabilistic finite state automata – part I and II. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(7):1013–1039, July 2005.
- [10] P. García and E. Vidal. Inference of K-testable languages in the strict sense and applications to syntactic pattern recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(9):920–925, 1990.
- [11] T. Dunning. Statistical identification of language. Technical Report CRL MCCS-94-273, Computing Research Lab, New Mexico State University, 1994.

CroMo – Morphological Analysis for Standard Croatian and its Synchronic and Diachronic Dialects and Variants¹

Damir ČAVAR ^{a,2}, Ivo-Pavao JAZBEC ^b and Tomislav STOJANOV ^b

^a *University of Zadar, Linguistics Dept., Croatia*

^b *Institute of Croatian Language and Linguistics, Zagreb, Croatia*

Abstract. CroMo is a finite state transducer that combines three major functionalities into one high-performance monolithic machine for morphological segmentation, annotation, and lemmatization. It is designed to be flexible, extensible, and applicable to any language that allows for purely morphotactic modeling on the lexical level of morphological structure. While it minimizes the development cycle of the morphology, its annotation schema maximizes interoperability by using a direct mapping from the GOLD ontology of linguistic concepts and features. The use of standardized ontology based annotations provides advanced possibilities for a DL-based post-processing of the annotated output.

Keywords. Croatian, Finite State Transducer, Morphology

Introduction

Quantitative and qualitative information about morphological properties of languages is hard to come by. For many languages information as for example contained in CELEX [1] is not available. For many research questions, most of the available information about distributional properties of morphemes and their feature makeup is not sufficient.

Corpus annotations tend to be lexeme and word-form oriented, providing part-of-speech (PoS) tags for tokens in the corpus, rather than segmentation of word-forms into morphemes and allomorphs with their particular feature annotation. The notion of *morphological information* is used inconsistently in the literature, e.g. associated with lexeme and PoS information only. The documented Croatian morphological lexicon [2] for example does not provide information about the morphological structure and specific feature annotations of single morphemes, but rather word-forms and lexemes with PoS-annotation.

¹Thanks to Thomas Hanneforth, Adrian Thurston, and Darko Veberič for their comments and help, and to our colleagues at the IHJJ for lexical material and linguistic advice, as well as several anonymous reviewers for helpful hints and comments.

²Corresponding Author: University of Zadar, Linguistics Dept., Obala kralja Petra Krešimira IV. 2, 23000 Zadar, Croatia; E-mail: dcavar@unizd.hr.

Table 1. Morphological parse example

token	popijemo		
parse	po	pije	mo
	stem		suffix
	prefix	root	inflectional
	aspect	verb	1 st
	perfective	transitive	plural

On the other hand, specific research questions, require detailed morphological analyses of lexical tokens in a corpus. In our particular case, the Croatian Language Corpus [3], as one of our major data sources needs to be annotated for subsequent analysis.

Our understanding of a morphological lexicon and morphological corpus annotation includes parsed lexemes on the morphological level, with annotations and explicit feature bundles associated with each single morpheme or allomorph, as shown in table 1 for the word *popijemo* (Croatian, “to drink (up)”).

In the first step we do not require hierarchical tree structure for morpheme relations, although it might be useful to reveal scope ambiguities of semantic properties. Thus the parses are just linear segmentations that include a quasi-hierarchical dependency with for example the prefix and root being contained in the stem, as shown in table 1. We are interested in all possible parses that lead to a complete analysis of a morphological complex word, i.e. all ambiguities. We do not intend to dissambiguate those at this stage.

Textual data from Croatian synchronic and diachronic dialects and variants is problematic, since e.g. different orthography standards have been and are still used. Looking at diachronic data another problem is that the lexical environment is not static, with lexical items emerging and disappearing, their semantic properties changing etc. Lexical changes occurred, some might have affected the morphological makeup of individual word-forms (including changes in paradigms), some might be related to different feature bundles associated with them.

Since various domains of lexical and morphological properties and features in our particular case are still subject to ongoing research, the set of features is necessarily open and unspecified from the outset. We expect in particular semantic properties, new feature types that result from linguistic conceptual necessities, or marking of linguistic origin and cultural background to emerge during future studies, i.e. the annotations of morphemes should be extensible.

Once the morphological segmentation is available, the generation of lemmata (lexical base-forms) can be achieved by appending the canonical inflectional suffix to the identified base, and potentially applying the necessary allomorphic change to the root. Furthermore, for establishing associations of word-forms to semantic fields, i.e. identifying the semantic root of a complex word-form, the lemma of the root provides a useful additional annotation information. For most Slavic and Germanic languages the rightmost root in a word-form is the semantic head of a complex morpheme. Thus, the root-lemma is generated by picking the rightmost root morpheme and append to it the canonical inflectional suffix. We annotate individual word forms for both lemma types, i.e. the root- and the base-lemma. The latter is achieved by inclusion of all prefixes in the lemma formation rule that are part of the morphological base.

The technical realization of the described annotator appears to be feasible, with a very simple, and nevertheless efficient technical solution, i.e. finite state transducers

[4,5]. In the following we describe the algorithmic specification of CroMo, the morphological parser, annotator and lemmatizer, developed for the Croatian standard, and synchronic and diachronic variants.

1. Previous approaches

Finite state methods for computational modeling of natural language morphology are wide-spread and well understood. Various commercial and open-source FSA-based development environments, libraries and tools exist for modeling of natural language morphology. A detailed discussion of their properties and application for various languages would be beyond the scope of this article. Some overview can be found in recent literature, e.g. [6,7,8], further links to literature and implementations can be found in the context of the OpenFst library [9].

For Croatian there are various descriptions of the formalization and computational modeling of morphology in terms of finite state methods [10,11]. However, an implemented testable application is not available.

Some solutions that have been implemented for example for German come close to the system requirements specified above. The SMOR [12] and Morphisto [13] systems partially represent such a type of computational morphology application. An almost complete overlap of features and properties can be found in the implementation of the German morphology as described in the TAGH [14] system.

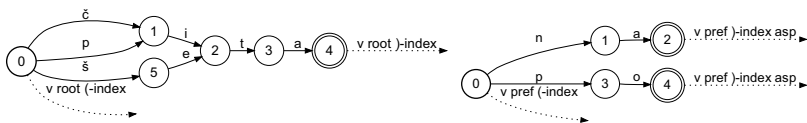
2. FST for morphological segmentation

For various reasons, we decide to stick to the approach and implementation strategy of TAGH, while we apply our own experimental libraries and development environment.³ Following the TAGH-approach [14], we model Croatian morphology by referring exclusively to morphotactic regularities, using morpheme and allomorph sets and regular morphological rules, such that a deterministic finite state transducer (FST) can be generated.

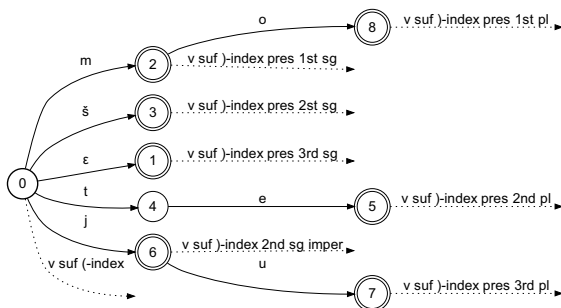
The initial modeling step is to group morphemes, identifying those on the basis of (a.) morphemes having the same feature specification, and (b.) being subject to the same morphological rules.

In CroMo each morpheme group represents one deterministic and acyclic finite state transducer (DFST). The design is similar to the Mealy [15] or Moore machine [16]. Every morpheme DFST emits on entry a tuple of the byte-offset in the input string, and the feature bundle that is associated with the DFSA path. In every final state the DFST emits the same tuple. This way morphemes are marked with a start and end index, as well as the corresponding feature bundle, representing the desired annotation. The following graph shows a simplified example of an acyclic DFST for verbal roots and for example aspectual prefixes:

³The automata and grammar definitions we use are compatible with several existing systems and libraries.



The verbal inflectional paradigm is organized in the same way. Since the model is based on purely morphotactic distributional regularities, potential phonological phenomena are expressed using exclusively allomorphic variations. The following graph shows a simplified network for verbal suffixes:



Once all morphemes are grouped into DFSTs, and the appropriate emission symbols (the annotations) are assigned to each entry and final state of the DFST, each morpheme group is assigned an arbitrary variable name, which is used in the definition of rules. A rule that makes use of the automata above could be defined as follows:

vAspectPref* . vAtiRoots . vInflSuf

This rule describes the concatenation of the verbal root DFST with the DFST for the verbal inflectional paradigm, using common regular expression notation. In this case we use the regular expression syntax as defined for the Ragel [17] state machine compiler. Additionally, the prefixes are defined as optional and potentially recursive prefixes concatenated with the verbal root DFST. This definition generates a cyclic⁴ deterministic transducer.

Such a DFST emits a tuple containing the byte-offset and the corresponding annotation symbols at the initial state, and at each morpheme boundary (former initial and final states of the sub-DFSTs).

⁴Cyclicity in this particular case leads to more compact automata. In principle, the depth of recursion of such prefixes could be limited (empirically and formally), and formalized using the appropriate regular expression syntax.

Using this approach, all lexical classes are defined as complex (potentially cyclic) DFSTs, and combined, together with the closed class items, as one monolithic DFST.

The advantage of such a representation is not only that the resulting morphological representation is compressed, but also that it is processed in linear time, with the identification of morpheme boundaries and corresponding feature bundles being restricted by contextual rules.

In order to cope with morphological ambiguity, this approach is extended. In principle there are two major approaches to deal with ambiguity, either one has to allow for non-deterministic automata (two different transitions with the same input emit a different output tuple), or ambiguity is mapped on the emission of multiple annotation tuples. In the case of CroMo, the latter option is used in the modeling. Every emission is a tuple of length 0 to n , such that e.g. orthographically ambiguous nominal suffixes like *a* (genitive singular or plural) are modeled as a single transition in a DFST with the final state emitting two annotation tuples that contain the specific case and number features.

2.1. Interoperability and annotation standard

Current language resources face a serious problem, related to issues of interoperability and annotation compatibility. Various different tag-sets are used for particular languages, and some of those tend not to be straight-forward compatible. In the same way, linguistic annotation tools do not necessarily make use of some standardized tag-set, and such a tag-set actually does not even exist.

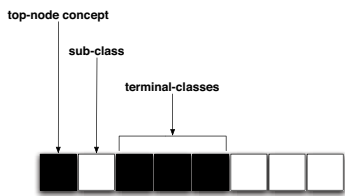
For our purposes here we decided to offer maximal interoperability in the resulting corpus annotation, as well as in the annotation tool as such, one that is maximally compatible with existing tag-sets, as language specific as necessary, and at the same time maximally extensible. The General Ontology for Linguistic Description (GOLD) [18,19,20] *was originally envisioned as a solution to the problem of resolving disparate markup schemes for linguistic data*. GOLD specifies basic linguistic concepts and their interrelations, and can be used, to a certain extent, as a description logic for linguistic annotation.

We make use of three core concept classes in GOLD, and the necessary sub-concepts, i.e. MorphoSemanticProperty, MorphosyntacticProperty, and LinguisticExpression. The concepts defined therein relate to the notions that are expected to be emitted by CroMo, i.e. morphological properties of morphemes (e.g. prefix, suffix, root), morpho-syntactic properties (e.g. case, number), and morpho-semantic properties (e.g. aspect, mood, tense).

By using the labels for concepts as defined in GOLD, we should be able to maintain maximal compatibility with other existing tag-sets. While the logic of GOLD would burden a morphological parsing algorithm, the reference to the concepts doesn't seem problematic. Representing the concepts as pure emission strings associated to the emission states, as discussed above, might decrease memory and performance benefits of a DFST-based analyzer. To maximize the performance, the GOLD-concepts and relations are mapped on a bit-vector. Encoding of the relevant concepts can be achieved with bit-vectors of less than 64 bit.

The mapping defines constants that correspond to bit-masks that are pre-compiled into the DFST. The bit-mask for example for *Genitive* might be defined as one that corresponds to set first and second bits of the terminal-class bit-field, plus the corre-

sponding bits that indicate that the sub-class `CaseProperty` is set, as well as the bit for the corresponding top-node class `MorphosyntacticProperty`, as shown in the following graphic:



In a limited way, via definitions of constants and mapping of linguistic annotation in the morpheme dictionaries, one can maintain implicatures and inheritance relations, as defined in the ontology, via bit-vector representations and appropriate bit-masks.

For the morphological analyzer this does not imply any additional processing load, i.e. the emission tuples consist of bit-vectors in form of 4-byte numerical integer values. Converting the emission tuples (i.e. individual bit-vectors) into literal string representations can be achieved efficiently, once an input string is analyzed completely.

2.2. Implementation

CroMo consists of two sets of code-bases. The first component converts a lexical base into a formal automaton definition. The second compiles together with the automaton definitions into a binary application.

The lexical base is kept either in database tables, spreadsheets, or textual form. The different formats allow us to maintain a minimally invasive lexical coding approach. Linguists or lexicologists are not required to learn a formal language for DFST definitions. Furthermore, they are free to use their individual way of annotation, being guided by GOLD concepts, but free to define their own, should these not be part of GOLD. CroMo provides guidelines for the data-format, but also the possibility to use individual scripts for data conversion and annotation mappings.

The individual morpheme lists, annotations and rule definitions are compiled into Ragel [17] automata definitions, as described above. Besides rules that are related to concrete morpheme lists and the corresponding DFSTs, there are also guessing rules that define general properties of nouns, verbs and adjectives. The features that are used, be they specified in GOLD or not, are mapped on bit-vectors, and C-header files with the constant literal and bit-vector mask definitions are generated.

Ragel generates a monolithic DFST as C-code, using highly efficient C-jump code (`goto`-statements), as well as a DOT-file for visualization of the resulting automaton (using e.g. Graphviz⁵). The generated code is wrapped in a C++ class that handles input and output, and controls the program logic.

In the current version the generation of the root- and the base-lemma is encoded in the emission bit-vector. One byte is reserved to mark the reverse offset for string concatenation, while two bytes are reserved to point to an element in a string array with the corresponding string that needs to be appended. The form *čitamo* would be associated

⁵See <http://www.graphviz.org/> for details.

with an offset of -2 and a corresponding suffix *ti*. This solution doesn't match the general paradigm, and is just temporary. In the next release the output characters of the corresponding lemma will be integrated in the emission of the transducer, associated with each single transition. Thus every emission will be a tuple that contains tuples of output characters and optional annotation bit-vectors.

CroMo expects a token list as input. Tokens are processed sequentially. For each token, all emitted tuples are collected in a stack. Only matching start- and end-tuples are returned, if there are compatible sub-morpheme analyses that span over the complete input token length.

The significant implementation features that differentiate CroMo from other solutions, are that the code-base is platform independent and open-source, based on free and open tools like GCC and Ragel. Furthermore, the fact that doesn't transcode the lexical base or the input words, it can be based on any encoding, even mixed encodings. The processing is purely binary (i.e. byte-oriented).

The extension of the morphological base is kept trivial, along the lines of the requirements specified above, i.e. the necessity to be able to add newly identified morphemes or paradigms from diachronic and synchronic variants.

3. Evaluation

The evaluation version of CroMo contains approx. 120,000 morphemes in its morpheme-base, using UTF-8 character encoding. The number of strings it can recognize is infinite, due to cyclic sub-automata. Unknown word-forms can be analyzed due to incorporated guessing rules.

For the following evaluation results we used a 2.4 GHz 64-bit Dual-Core CPU. In the evaluation version only a single core is used during runtime of the FST, while both CPU cores are used during compilation.

Compilation of the morphology requires min. 4 GB of RAM using GCC 4.2. This is expected due to the monolithic architecture, and since the Ragel-generated C-code of the transducer gets very large. The compilation process takes less than 5 minutes, using both CPU cores. The resulting binary footprint is less than 5 MB of size.

The final automaton consists of approx. 150,000 transitions and 25,000 states.

We selected randomly 10,000 tokens with an average morpheme length of 2.5 morphemes. CroMo processes in average approx. 50,000 tokens per second (real 10,000 tokens per 150 millisec.), including runtime instantiation in memory, mapping of the analysis bit-vectors to the corresponding string representations, generation of lemmata, and output redirection to a log-file. An extension of the morpheme base has no significant impact on memory instantiation time, neither on the runtime behavior. The memory instantiation can be marginalized for a large processing sample.

CroMo doesn't implement transitional or emission-probabilities, due to missing quantitative information from training data. Once an annotated corpus is available, these weights can trivially be implemented as additional weights in the emission tuple.

A relevant evaluation result is the coefficient of the ratio between all and relevant emissions, i.e. the percentage of relevant (possible) morpheme analyses and all generated ones. Due to certain limitations, we did not perform such an evaluation, neither a recall evaluation on a predefined evaluation corpus. The results of these eval-

uations, together with the source code, will be made available on CroMo's web site <http://personal.unizd.hr/~dcavar/CroMo/>.

References

- [1] Gavin Burnage. CELEX - A guide for users. Technical report, Centre for Lexical Information, University of Nijmegen, Nijmegen, 1990.
- [2] Antoi Oliver and Marko Tadić. Enlarging the croatian morphological lexicon by automatic lexical acquisition from raw corpora. In *Proceedings of LREC 2004*, volume IV, pages 1259–1262, Lisbon, May 2004. ELRA.
- [3] Dunja Brozović-Rončević and Damir Čavar. Hrvatska jezična riznica kao podloga jezičnim i jezičnopovijesnim istraživanjima hrvatskoga jezika. In Vidjeti Ohrid, editor, *Hrvatska sveučilišna naklada*, pages 173–186, Zagreb, 2008. Hrvatsko filološko društvo.
- [4] Jean Berstel. *Transductions and Context-Free Languages*. Teubner Studienbücher, Stuttgart, 1979.
- [5] Jean Berstel and Christophe Reutenauer. *Rational Series and Their Languages*. EaTCS Monographs on Theoretical Computer Science. Springer-Verlag, Berlin, December 1988.
- [6] Richard Sproat. *A Computational Theory of Writing Systems*. AT&T Bell Laboratories, New Jersey, July 2000.
- [7] Kenneth R. Beesley and Lauri Karttunen. *Finite State Morphology*. CSLI Publications, Stanford, April 2003.
- [8] Brian Roark and Richard Sproat. *Computational Approaches to Syntax and Morphology*. Oxford University Press, Oxford, 2007.
- [9] Cyril Allauzen, Michael Riley, Johan Schalkwyk, Wojciech Skut, and Mehryar Mohri. OpenFst: A general and efficient weighted finite-state transducer library. In *Proceedings of the Ninth International Conference on Implementation and Application of Automata*, (CIAA 2007), pages 11–23. Springer-Verlag, 2007.
- [10] Marko Tadić. *Računalna obradba morfologije hrvatskoga književnog jezika*. PhD thesis, Filozofski fakultet Sveučilišta u Zagrebu, Zagreb, Croatia, 1994.
- [11] Vjera Lopina. *Strojna obrada imenične morfologije u pisanome hrvatskom jeziku*. Master's thesis, Centar za postdiplomske studije Dubrovnik, Dubrovnik, October 1999.
- [12] Helmut Schmid, Arne Fitschen, and Ulrich Heid. SMOR: A German computational morphology covering derivation, composition, and inflection. In *Proceedings of the IVth International Conference on Language Resources and Evaluation (LREC 2004)*, pages 1263–1266, Lisbon, Portugal, 2004.
- [13] Andrea Zielinski and Christian Simon. Morphisto – an open-source morphological analyzer for German. In *Proceedings of FSMNLP 2008*, Ispra, Italy, September 2008.
- [14] Alexander Geyken and Thomas Hanneforth. TAGH: A complete morphology for german based on weighted finite state automata. In Anssi Yli-Jyrä, Lauri Karttunen, and Juhani Karhumäki, editors, *FSMNLP 2005*, volume 4002 of *Lecture Notes in Artificial Intelligence*, pages 55–66. Springer, September 2005.
- [15] George H. Mealy. A method for synthesizing sequential circuits. *Bell System Technical Journal*, 34(5):1045–1079, September 1955.
- [16] Paul E. Black. Dictionary of algorithms and data structures. Online publication: U.S. National Institute of Standards and Technology, Available from <http://www.nist.gov/dads/HTML/mooreMachine.html>, December 2004.
- [17] Adrian D. Thurston. Parsing computer languages with an automaton compiled from a single regular expression. In *11th International Conference on Implementation and Application of Automata (CIAA 2006)*, volume 4094 of *Lecture Notes in Computer Science*, pages 285–286, Taipei, Taiwan, August 2006.
- [18] Scott O. Farrar and D. Terence Langendoen. A linguistic ontology for the semantic web. *Glott International*, 7(3):1–4, March 2003.
- [19] Scott O. Farrar, William D. Lewis, and D. Terence Langendoen. A common ontology for linguistic concepts. In N. Ide and C. Welty, editors, *Semantic Web Meets Language Resources: Papers from the AAAI Workshop*, pages 11–16, Menlo Park, CA, 2002. AAAI Press.
- [20] Scott O. Farrar. *An Ontology for Linguistics on the Semantic Web*. PhD thesis, The University of Arizona, Tucson, Arizona, 2003.

Forest FIRE and FIRE Wood: Tools for Tree Automata and Tree Algorithms

Loek CLEOPHAS ^{a,1}

^a *Software Engineering & Technology Group,
Department of Mathematics and Computer Science,
Eindhoven University of Technology,
P.O. Box 513, NL-5600 MB Eindhoven, The Netherlands*

Abstract. Pattern matching, acceptance, and parsing algorithms on node-labeled, ordered, ranked trees ('tree algorithms') are important for applications such as instruction selection and tree transformation/term rewriting. Many such algorithms have been developed. They often are based on results from such algorithms on words or generalizations thereof using finite (tree) automata. Regrettably no coherent, extensive toolkit of such algorithms and automata existed, complicating their use.

Our toolkit FOREST FIRE contains many such algorithms and automata constructions. It is accompanied by the graphical user interface (GUI) FIRE WOOD. The toolkit and GUI provide a useful environment for experimenting with and comparing the algorithms. In this tool paper we give an overview of the toolkit and GUI, their context and design rationale, and mention some results obtained with them.

Keywords. tree automata, tree algorithms, toolkit, GUI, instruction selection

Introduction

Pattern matching, acceptance, and parsing algorithms ('tree algorithms') are important for applications such as instruction selection and tree transformation/term rewriting. In instruction selection for example, an intermediate representation tree for a program fragment needs to be covered using tree patterns, each of which corresponds to a CPU instruction. (For example, tree pattern $+(R_i, c)$ might correspond to instruction *ADD* R_i, c —the addition of constant c to the value in register R_i . A cover of an intermediate representation tree using such patterns then yields an instruction sequence for the program fragment.)

Many tree algorithms (on node-labeled, ordered, ranked trees) appeared in the literature [1,2,3,4,5,6,7,8,9,10,11,12,13]. They are usually based on results from such algorithms on words or generalizations thereof, but are often scattered over the literature and lacking reference to the underlying theory. A few good overview works exist, but they are focused on theory instead of algorithms [14,15,16]. Together with the lack of a coherent and extensive toolkit, this complicated comparison of and choice among them.

¹E-mail: loek@loekcleophas.com.

TABASCO [17,18] was therefor applied to the domain of tree algorithms. TABASCO (TAXonomy-BASEd Software CONstruction) is a *domain modeling* and *domain engineering* method for algorithmic domains. It brings order to and increases accessibility of a specific domain by taxonomizing algorithms and creating a *domain specific toolkit* based on the resulting taxonomy.

A literature survey is first performed to find algorithms from a domain. A taxonomy is then constructed, forming a domain model by classifying algorithms—both from the literature or based on variations of those from the literature—according to essential details, somewhat similar to a biological taxonomy.

By indicating commonalities and differences between the various algorithms, a taxonomy simplifies the design of a coherent toolkit. High-level design choices are guided by the taxonomy structure, while language constructs to implement smaller design parts can be chosen using standard design techniques. Furthermore, the taxonomy presentation of algorithms helps in implementing them.

Our taxonomies for tree acceptance and tree pattern matching are presented in [19, Chapters 5 and 6]. The two problems are related and algorithms solving them involve related algorithmic ingredients. The commonalities lead to similarities between the taxonomies constructed.

The similarities were used in the taxonomy-based toolkit design and implementation of FOREST FIRE. It contains three kinds of components: tree acceptance and pattern matching algorithms (based on the taxonomies of such algorithms; additionally a few tree parsing algorithms are included), representations of and constructions for the tree automata used in these algorithms, and basic supporting data structures and algorithms. Apart from providing experience with taxonomy-based toolkit design, FOREST FIRE is useful for experiments aimed at understanding and comparing the many algorithms and automata. A GUI, FIRE WOOD, was constructed to facilitate this and show the toolkit's usability.

Our toolkit and GUI offer many different tree automata types and construction, acceptance, matching, and parsing algorithms. This large collection of implementations sets them apart from all other tree tools we know of.

The toolkit and GUI were implemented in Java, using the Standard Widget Toolkit (SWT), for use on Apple Mac OS X, Linux, and Microsoft Windows XP, and will be made available in the near future. In total, they contain about 140 classes and 16 thousand lines of code.

In this tool paper we give a brief high-level overview of the toolkit and GUI and mention some results obtained with them.

1. Related Work

Related work on taxonomy-based toolkits and TABASCO is discussed in [17].

Instruction selection tools each use a *single* tree parsing algorithm and automaton type. Such tools include BEG [20], BURG/ iBURG [21], and TWIG [1].

Quite a few toolkits and GUIs use (ordered, ranked) trees and tree automata for other applications. Such toolkits and GUIs generally also contain just one or a few automaton representations and algorithm variants, and their focus is on using such representations and algorithms in their particular application area.

Timbuk [22] uses tree automata for reachability analysis in term rewriting. It represents terms by automata and uses operations on such automata, which are implemented by the underlying Taml library. Taml and Timbuk include a number of operations on tree automata (boolean operations, determinization, decision problems, matching, input/output). They are complemented by Tabi, a GUI to interactively construct trees and explore the state assignments done to them.

Tiburon [23] is a toolkit based on weighted top-down tree automata and aimed at natural language processing. It contains weighted top-down acceptors and transducers, and e.g. construction of such automata based on regular tree grammars, boolean operations, and weighted determinization.

MONA [24] uses binary trees and corresponding tree automata to solve decision problems in the weak monadic second-order theory of 2 successors. It uses bottom-up tree automata with an initial top-down pass to restrict the state space of each tree node and thereby prevent state space explosions; in our toolkit, such explosions are prevented a.o. by filtering techniques (see Section 2).

Finally, TREEBAG [25] contains various kinds of tree grammars and tree transducers and allows them to be composed to generate and transform trees, but does not include automata constructions or pattern matching algorithms.

2. Toolkit

The FOREST FIRE toolkit contains the three kinds of components mentioned in the introduction.

Basic supporting data structures and algorithms include e.g. symbols (*terminals* of fixed arity and *variables* or *nonterminals*, both of arity 0), alphabets, tree nodes (including a symbol and links to parent and children), trees, regular tree grammars (consisting of an alphabet, a *start symbol* nonterminal, *productions* allowing the replacement of a nonterminal by a tree whose leafs may be labeled by nonterminals), and tree pattern sets (sets of trees over an alphabet including *variables*), as well as algorithms for analyzing trees, regular tree grammars, and productions (such as analyzing whether a grammar contains *chain rules*), and algorithms for transforming them (such as those removing chain rules using a transitive closure computation). All of these are required or useful as basic building blocks for the implementation of tree acceptance and tree pattern matching algorithms and accompanying automata constructions.

Automata are essential to the tree acceptance and pattern matching algorithms from the taxonomies. FOREST FIRE implements a number of (tree) automata types, i.e. non-deterministic tree automata with and without ε -transitions, deterministic ones, and deterministic word automata of the Aho-Corasick type. The tree automata can be of types assigning states to trees frontier-to-root or root-to-frontier, while the Aho-Corasick automata are used root-to-frontier. Deterministic frontier-to-root tree automata using *filtering* are also included.²

For lack of space, automata construction details are not discussed, but tree automata form generalizations from the word case, and many of the concepts are similar—for example, states have a relation to (parts of) regular tree grammars or tree patterns, similar

²Filtering is a well known technique to reduce the memory use of a tree automaton's transition tables by using properties of the trees underlying its states [3].

to the relation between states and regular grammar/expression elements in the word case. By using different state sets obtained from a regular tree grammar or tree pattern set, different automata of each of the types mentioned can be constructed. In total, FOREST FIRE includes over thirty tree acceptor and tree pattern matcher constructions.

Despite the many constructions, just a few acceptance and pattern matching algorithms—each using an automaton resulting from a construction—needed to be implemented. For e.g. pattern matching, four were implemented, using nondeterministic root-to-frontier and frontier-to-root tree automata, using deterministic frontier-to-root ones, and using Aho-Corasick automata respectively.

The abstract algorithms from the taxonomies were translated into Java code in a straightforward way. The coherence and factoring of commonalities—of states, behavior, interface, operations—in the taxonomies and the similarities between them made toolkit design and implementation simple and easy. This is in line with earlier experience with taxonomy-based software construction [17,18].

3. GUI

FIRE WOOD is a GUI accompanying FOREST FIRE, facilitating input, output, creation and manipulation of data structures from the toolkit. It supports interactive experiments involving grammar transformations, automata constructions, acceptance, pattern matching, and parsing.

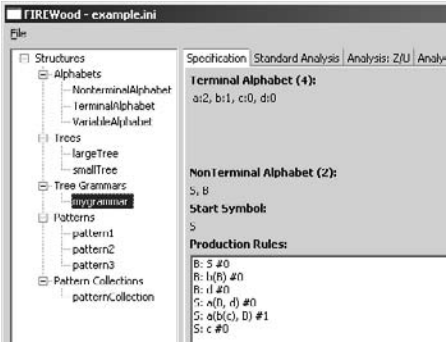


Figure 1. Example showing the specification of a tree grammar.

To start working with FIRE WOOD a user loads a file with definitions of alphabets, trees, grammars, pattern sets etc. These definitions are shown in a tree view, as on the left of Figure 1. For each structure, various operations—depending on the structure’s type—may be available, e.g. for a pattern set a pattern matching automaton can be created and used in a pattern matching algorithm (applied to a tree set also taken from the input file). The operations on each structure are accessed using the tab pages that appear to the right of the tree view upon selection of a structure.

The tabs fall into just a few categories:

- Specification tabs for structures. Figure 1 shows this tab for a regular tree grammar generating trees like c (using the last production) and $a(b(c), d)$ (using the fifth and the third production).

- Analysis tabs for tree grammars, providing statistics e.g. about chain rules.
- Transformation tabs for tree grammars, providing access to transformations such as removal of (all or selected) chain rules.
- Automaton construction tabs, allowing automata of a specific type to be constructed for a tree grammar or pattern set, using construction specific settings. Construction time, memory usage, automaton size, and the automaton's structure are reported following construction. Figure 2 shows an example of such a tab for the tree grammar case.
- Acceptance/matching algorithm tabs, allowing such algorithms to be applied to a grammar and tree(s) or to a pattern set and tree(s) respectively. Based on the selected automaton's type, the appropriate acceptance/matching algorithm is automatically selected. This algorithm is run on the tree(s) selected by the user, returning a boolean for each tree (in the case of an acceptance algorithm) or a set of matching patterns for every node of each tree (in the case of a matching algorithm). As for automaton construction tabs, data related to benchmarking is provided.

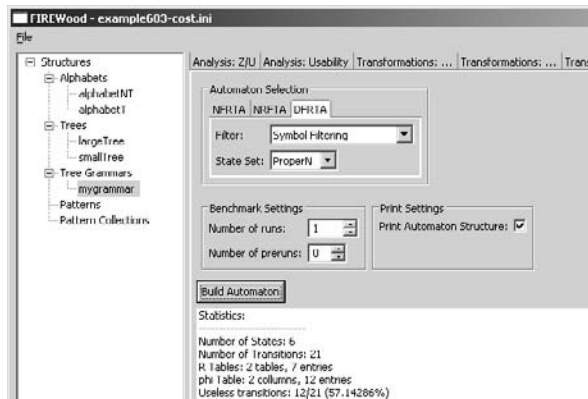


Figure 2. Example deterministic frontier-to-root tree automaton construction.

4. Practical Results

The practical results fall into two categories: those for the toolkit and GUI themselves—particularly experiences with design, implementation, and extensibility—and those for the algorithms and constructions contained in them—particularly in terms of experimental results obtained from benchmarking them.

The taxonomy-based design and implementation of the toolkit were beneficial in multiple ways: as with earlier taxonomies [17,18], the taxonomies—with their uniform algorithm descriptions and explicit factoring of algorithms' commonalities and differences—made toolkit design and implementation quite straightforward [19,26]. Furthermore, the coherent, taxonomy-based design made the toolkit easily extendable: two tree parsing algorithms—not part of the taxonomies but straightforward extensions of

tree acceptance algorithms—were added in less than two hours with just tens of lines of code.

A number of experiments was performed with FOREST FIRE and FIRE WOOD: with tree grammar transformations, with different (tree) automata constructions for use in tree acceptance and tree pattern matching algorithms, and with these algorithms using the different automata constructed.

Automata construction experiments for pattern matching and acceptance used various pattern sets and tree grammars. Examples ranged from small ‘toy’ examples to a rewrite system taken from a model transformation case study and to grammars used for instruction selection for e.g. the Intel X86. Experiments with pattern matching and acceptance algorithms using the automata thus constructed were also performed.

The most interesting results obtained are those on *deterministic frontier-to-root tree automata with filtering*. These have been used in e.g. BURG [21]. BURG uses a well known filter originally by Chase [3] to (drastically) reduce memory use compared to unfiltered automata. We showed that Chase’s technique was an instance of a more general technique and that another filter, TFILT, had appeared (somewhat implicitly) in the literature before [9]. More importantly, we described two new, simpler filters. We compared construction times and memory use of resulting automata using the various filters. As an example, the results for the Intel X86 instruction selection grammar (mentioned above) are displayed in Table 1 below. Although the results depend on the pattern set/grammar used, for instruction selection grammars such as this one the new IFILT consistently outperforms the others—including Chase’s existing CFILT—in memory use of the resulting automata, while the new SFILT does so in construction time.

Table 1. Automata constructions including ones with filtering.

Filter	None	None, reduced state set	TFILT	SFILT	IFILT	CFILT
Memory usage (MiB)	144.3	2.3	2.5	4.6	1.4	10.1
Construction time (ms)	273009	1626	7398	267	2883	288

(Detailed) results on all the experiments can be found in [19,26].

5. Concluding Remarks

We gave a brief impression of the FOREST FIRE toolkit and FIRE WOOD GUI. We also discussed their context and design rationale, and mentioned some interesting results obtained by experimenting with them. The toolkit has already been applied to tree parsing, by extending some of the tree acceptance algorithms with little effort [26]. To simplify use of algorithms from the toolkit by users that are not domain experts, a *domain specific language* could be developed. Such a language would allow a user to automatically obtain an algorithm by specifying some parameter values (e.g. importance of memory use vs. preprocessing/automaton construction time, direction of tree processing).

We also plan to apply the toolkit to instruction selection and tree transformation/term rewriting. In the latter, the combination of different pattern matching algorithms and

rewriting strategies—particularly others than the usual leftmost innermost—would be investigated. Particularly interesting would be to see how match results on a tree can be efficiently updated following a single rewrite step, and whether certain rewriting strategies seem more useful or efficient than others in term rewriting in the domain of model transformations.

For instruction selection, the application of the new filters instead of Chase’s filter is of interest. An instruction selector can be implemented in Java using the toolkit, or one or more of the new filters can be implemented in C++ as part of BURG [21], which uses Chase’s filter.

More details on the software are available in [19, Chapter 8] and in [26]. The FOREST FIRE and FIRE WOOD software and manuals will be available at <http://www.fastar.org> and <http://www.win.tue.nl/set>.

Acknowledgments

I thank Roger Strolenberg for his work on the toolkit and GUI, and for the experiments performed with them, during his Master’s thesis research and a subsequent temporary appointment in our research group. Mark van den Brand, Bruce Watson, Kees Hemerik and the referees provided valuable comments.

References

- [1] A. V. Aho, M. Ganapathi, and S. W. K. Tjiang. Code generation using tree matching and dynamic programming. *ACM Transactions on Programming Languages and Systems*, 11(4):491–516, 1989.
- [2] A. Balachandran, D. M. Dhamdhere, and S. Biswas. Efficient retargetable code generation using bottom-up tree pattern matching. *Computer Languages*, 15(3):127–140, 1990.
- [3] David R. Chase. An improvement to bottom-up tree pattern matching. In *Conference Record of the Fourteenth Annual ACM Symposium on Principles of Programming Languages*, pages 168–177. ACM, 1987.
- [4] Christian Ferdinand, Helmut Seidl, and Reinhard Wilhelm. Tree automata for code selection. *Acta Informatica*, 31:741–760, 1994.
- [5] Philip J. Hatcher and Thomas W. Christopher. High-quality code generation via bottom-up tree pattern matching. In *Conference Record of the Thirteenth Annual ACM Symposium on Principles of Programming Languages*, pages 119–130. ACM, 1986.
- [6] C. Hemerik and J. P. Katoen. Bottom-up tree acceptors. *Science of Computer Programming*, 13(1):51–72, 1989.
- [7] C. M. Hoffmann and M. J. O’Donnell. Pattern matching in trees. *Journal of the ACM*, 29(1):68–95, January 1982.
- [8] H. Kron. *Tree templates and subtree transformational grammars*. PhD thesis, University of California, Santa Cruz, 1975.
- [9] Prescott K. Turner. Up-down parsing with prefix grammars. *SIGPLAN Notices*, 21(12):167–174, December 1986.
- [10] H. J. A. van de Meerakker. Een parsing algoritme voor boomgrammatica’s. Master’s thesis, Faculteit Wiskunde en Informatica, Technische Universiteit Eindhoven, May 1988. (In Dutch).
- [11] Yolanda van Dinther. De systematische afleiding van acceptoren en ontleders voor boom-grammatica’s. Master’s thesis, Faculteit Wiskunde en Informatica, Technische Universiteit Eindhoven, August 1987. (In Dutch).
- [12] Beatrix Weisgerber and Reinhard Wilhelm. Two tree pattern matchers for code selection. In Dieter Hammer, editor, *Compiler Compilers and High Speed Compilation, 2nd CCHSC Workshop, Berlin GDR, October 10-14, 1988, Proceedings*, volume 371 of *Lecture Notes in Computer Science*, pages 215–229, 1989.

- [13] R. Wilhelm and D. Mauer. *Compiler Design*. Addison-Wesley, 1995.
- [14] Hubert Comon, Max Dauchet, Rémi Gilleron, Florent Jacquemard, Denis Lugiez, Sophie Tison, and Marc Tommasi. Tree automata: Techniques and applications. TATA Website at <http://www.grappa.univ-lille3.fr/tata/>, 2007.
- [15] Ferenc Gécseg and Magnus Steinby. *Tree Automata*. Akadémiai Kiadó, Budapest, 1984.
- [16] Ferenc Gécseg and Magnus Steinby. *Tree Languages*, volume 3 of *Handbook of Formal Languages*, pages 1–68. Springer, 1997.
- [17] Loek Cleophas and Bruce W. Watson. Taxonomy-based software construction of SPARE Time: a case study. *IEE Proceedings Software*, 152(1), February 2005.
- [18] Loek Cleophas, Bruce W. Watson, Derrick G. Kourie, Andrew Boake, and Sergei Obiedkov. TABASCO: Using concept-based taxonomies in domain engineering. *South African Computer Journal*, 37:30–40, December 2006.
- [19] Loek G. W. A. Cleophas. *Tree Algorithms: Two Taxonomies and a Toolkit*. PhD thesis, Department of Mathematics and Computer Science, Eindhoven University of Technology, April 2008.
- [20] H. Emmelmann, F.-W. Schröer, and L. Landwehr. Beg: a generator for efficient back ends. *SIGPLAN Notices*, 24(7):227–237, 1989.
- [21] Todd A. Proebsting. Burs automata generation. *ACM Transactions on Programming Languages and Systems*, 17(3):461–486, 1995.
- [22] T. Genet and V. Viet Triem Tong. Reachability analysis of term rewriting systems with timbuk. In *Proceedings of the 8th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, volume 2250 of *Lecture Notes in Artificial Intelligence*. Springer, 2001.
- [23] Jonathan May and Kevin Knight. Tiburon: A weighted tree automata toolkit. In *Proceedings of the 11th International Conference on Implementation and Application of Automata*, 2006.
- [24] Nils Klarlund and Anders Møller. *Mona Version 1.4 User Manual*. Brics, Department of Computer Science, University of Aarhus, January 2001. Notes Series NS-01-1. Available from <http://www.brics.dk/mona/>. Revision of BRICS NS-98-3.
- [25] Frank Drewes. The TREEBAG homepage. <http://www.informatik.uni-bremen.de/theorie/treebag/>, Version 1.2, last update August 30 2001.
- [26] Roger Strolenberg. ForestFIRE & FIREWood, A Toolkit & GUI for Tree Algorithms. Master's thesis, Department of Mathematics and Computer Science, Eindhoven University of Technology, June 2007.

An XML Format Proposal for the Description of Weighted Automata, Transducers and Regular Expressions

Akim DEMAILLE ^a, Alexandre DURET-LUTZ ^a, Florian LESAIN ^a,
Sylvain LOMBARDY ^b, Jacques SAKAROVITCH ^c and Florent TERRONES ^a

^a *LRDE, EPITA, {name}@lrde.epita.fr*

^b *IGM, Université Paris Est Marne-la-Vallée, lombardy@univ-mlv.fr*

^c *LTCI, CNRS / ENST, sakarovitch@enst.fr*

Abstract. We present an XML format that allows to describe a large class of finite weighted automata and transducers. Our design choices stem from our policy of making the implementation as simple as possible. This format has been tested for the communication between the modules of our automata manipulation platform Vaucanson, but this document is less an experiment report than a position paper intended to open the discussion among the community of automata software writers.

Keywords. XML format, finite automata, weighted automata, transducers, regular expressions

Introduction

The aim of an interchange format for automata is to make possible, and hopefully easy, the communication between the various programs that input or output such objects.

There exist many kinds of (finite) automata: automata on finite words or on infinite words, automata on tuples of words (often called transducers), weighted automata where the weights can be taken in very different semirings, timed automata, counter automata, pushdown automata, Petri nets, *etc.* The scope of our proposal is restricted to *weighted automata and transducers on finite words*. These automata already form a large family and cover most of the needs in Finite State Machines that are relevant to Natural Language Processing.

To our knowledge, there does not exist any format representing this class of automata. Many tools have devised their own format for reading and writing automata. For instance Grail [1], FSM [2], OpenFST [3] each have their own textual representations of automata. Such representations, often integer-based, are concise and simple to parse but they are dedicated to one program, and will hardly allow any generalization. What if the weights of our automaton are not integers, or if we want to label the transition of an automaton with rational expressions instead of letters? Other formats, such as GraphML [4], are more generic and allow to represent any kind of graph, but they do not allow to represent the semantics associated to the automaton: indeed exchanging automata requires some typing information to be conveyed along with the structure.

Most of the design choices for our proposal for an exchange format have been shaped by the policy of making its implementation as simple as possible. This is already true of the option of choosing XML as the language for describing the format. Our proposal, called FSM XML, already covers a large class of automata but should be considered as a skeleton that can be completed to cater for other needs. We believe such a generic interchange format, should be of interest to the community.

FSM XML has been implemented and tested within Vaucanson [5], our automata manipulation platform, where it serves as an exchange format between components such as the core and the command-line interface. But this document is less a report on an experiment than a *position paper* intended to open the discussion.

Because of size constraints in these proceedings, we only give a brief summary of the FSM XML format and of the choices we have taken. We refer interested readers to our web page for more exhaustive information [6].

1. FSM XML Overview

We assume the reader familiar with the terminology of automata theory [7].

1.1. Data to Carry

The complete description of an automaton involves four different types of data: (1) the type of the labels, which amounts to define a semiring of series, a mathematical structure; (2) the automaton structure itself, that is a labeled graph; (3) if the automaton is to be seen on a screen or drawn in a figure, geometric data that tell where the states are located, and possibly the shape of the transitions, the relative location of its label; (4) finally, data which we call drawing data and that tell how the states and transitions are actually drawn, their size, the thickness the lines, the color, *etc.*

The two latter types are relevant only to applications that display the automaton in some way: they have no influence on the structural meaning of the automaton. Their presence is optional and we will focus on the first two items.

1.2. Automaton Description

An FSM XML description of an automaton consists in a tag `<automaton/>` containing two required children:

```
<automaton name='Example Automaton' readingDir=right>
  <valueType>...</valueType>
  <automStruct>...</automStruct>
</automaton>
```

The tag `<valueType/>` specifies the type of the labels of the automaton, and the tag `<automStruct/>` holds the description of the structure of the automaton (list of states and transitions with their labels).

The attribute `name` names the automaton and the attribute `readingDir` tells whether the automaton reads the word from left to right or from right to left.

1.3. The Labels: That Is the Question

The question of labels has three levels: (1) what are the types of labels that need to be supported? (2) how will these types be represented in the format? (3) how will the label of a given transition in an automaton will be represented in the format? The first commands over the two others.

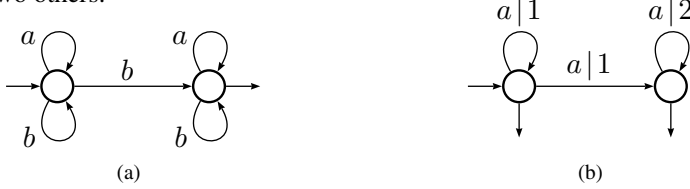


Figure 1. How many automata are there?

Let us consider Figure 1. At (a), we see an automaton whose labels are letters a and b and which clearly recognizes the set of words containing at least one b ; but we can consider that the same automaton is an automaton with multiplicity in \mathbb{N} where the coefficient of every transition is 1, in which case the automaton realizes the series which associates every word with its number of occurrences of b . At (b), we see an automaton which is clearly a weighted automaton, and the weights are (positive) integers, but we do not know without further information which is the *semiring structure* which is applied to these integers: it may be the ‘classical structure’, in which case the weight of a^n is 2^n , or a ‘tropical structure’, and the weight of a^n is n if we are in $(\mathbb{N}, \min, +)$, or $2n - 1$ if we are in $(\mathbb{N}, \max, +)$. This example makes clear that the description of an automaton *must* contain the definition of the *semiring of series* to which the behaviour of the automaton belongs. The requirement of such a strong typing is what distinguishes FSMXML from more general graph representation formats such as GraphML [4].

1.3.1. Label Types

We represent automata either over *free monoids* or over *products* of free monoids. Products of *arbitrary* number of free monoids allow to represent k -tape automata, that is, generalization of transducers that are 2-tape automata (we thus do not use the name ‘transducer’ in the description of the format).

The generators of the free monoids are either *simple* letters or *tuples* of simple letters of arbitrary dimension. The simple letters refers to simple types in programming languages such as *characters* (or subsets of them such as *letters* or *digits*) or even *integers*¹. Tuples of letters consist then in ordered sets of simple letters, not necessarily all of the same type: for instance, pairs of letter and digit will naturally represent indexed letters. Another very useful example: automata over free monoids whose generators are pairs of letters are equivalent to transducers which realize length preserving relations and are also used, modulo some technicalities, to represent *synchronized transducers*.

Within FSMXML, we represent *weighted* automata; the weights being taken either in *numerical semirings* or in *series semirings*. By ‘numerical semirings’, we refer to simple types of numbers, as they are implemented in any programming languages, like integers, or reals, together with conventional operations, or ‘unconventional’ ones that can be overloaded on the conventional ones. By ‘series semirings’, we refer to semirings

¹Automata used in the study of numeration systems, for instance, make use of labels that are integers.

that can be recursively defined by using the already defined numerical semirings and the (products of) free monoids. The reason for opening the possibility in FSM XML is Kleene-Schützenberger Theorem for transducers which states that a (finite) automaton over the product say $A^* \times B^*$ with multiplicity say in \mathbb{N} is equivalent to an automaton over A^* with multiplicity in the semiring of (rational) series over B^* with multiplicity in \mathbb{N} .

1.3.2. Label Type Representation

The three main features of label types that we want to represent are then: product of an *arbitrary number* of free monoids, generators that are vectors of *arbitrary dimension*, and *recursive definition* for semiring of multiplicities. These are easily and naturally taken into account in an XML format although it may end rapidly into rather long and apparently complicated description files.

Figure 2 shows three excerpts of FSM XML files that describe label types: the first one for a classical Boolean automaton over a two-letter alphabet, and the two others for the aforementioned two equivalent forms of transducers with multiplicities.

One understands that in both tags `<semiring/>` and `<monoid/>`, we use the attribute `type` to control the *syntax* of the tag: we call such attributes *pivotal*. In `<semiring/>`, `type = numerical` calls for two other attributes, `set` and `operation` that will be given *token* values, that is, conventional strings that have to be recognized and correctly interpreted by the parser. Whereas `type = series` calls for another succession of `<semiring/>` and `<monoid/>` tags.

In `<monoid/>`, `type = free` calls for the attributes `genKind`, `genDescrip` and `genSort`, whereas `type = product` calls for the attribute `prodDim`, an integer strictly larger than 1 which tells how many children `<monoid/>` this tag `<monoid/>` will have.

The attribute `genKind` is another pivotal attribute which controls whether the generators are *simple* or *tuple*. In the former case, `genSort` is a token which tells which kind of generators are expected: *letter*, *digit*, *alphanum*, or *integer*. The latter case is not exemplified in Figure 2 but the mechanism is of the same type as for product of monoids, although it is not recursive.

The attribute `genDescrip = enum` tells that the generators of the free monoid will be enumerated, by means of the attribute `value` in tags `<monGen/>`. The assignments `genDescrip = range` and `genDescrip = set` are meant to give way to the description of large alphabets such as those that are used in NLP. The precise syntax and semantic open by these tokens have still to be defined.

This constrained specification of the type of an automaton makes it easier to extend the format to support new types without redefining the complete semantics of each new automaton type. For instance to support automata over a log-probability semiring we would just have to introduce some new tokens (and their semantics) for the `set` or `operation` attributes.

1.3.3. Rational Expressions for Label Representation

It is quite a natural idea to be able to describe rational (that is, regular) expressions within an XML format for automata. The behaviour of a finite automaton over any monoid can be denoted by a rational expression, and most of the automata related software deal with the conversion between automata and expressions, back and forth.

```

<valueType>
  <semiring type=numerical set='B' operation='classical' />
  <monoid type=free genKind=simple genDescrip='enum' genSort='letter'>
    <monGen value='a' />
    <monGen value='b' />
  </monoid>
</valueType>

```

(a) Type for a Boolean automaton over $\{a, b\}^*$.

```

<valueType>
  <semiring type=numerical set='N' operation='classical' />
  <monoid type=product prodDim='2'>
    <monoid type=free genKind=simple genDescrip='enum'
genSort='letter'>
      <monGen value='a' />
      <monGen value='b' />
    </monoid>
    <monoid type=free genKind=simple genDescrip='enum'
genSort='letter'>
      <monGen value='a' />
      <monGen value='b' />
    </monoid>
  </monoid>
</valueType>

```

(b) Type for an automaton over $\{a, b\}^* \times \{a, b\}^*$ with multiplicity in \mathbb{N}

```

<valueType>
  <semiring type=series>
    <semiring type=numerical set='N' operation='classical' />
    <monoid type=free genKind=simple genDescrip='enum'
genSort='letter'>
      <monGen value='a' />
      <monGen value='b' />
    </monoid>
  </semiring>
  <monoid type=free genKind=simple genDescrip='enum' genSort='letter'>
    <monGen value='a' />
    <monGen value='b' />
  </monoid>
</valueType>

```

(c) Type for an automaton over $\{a, b\}^*$ with multiplicity in $\mathbb{N}\langle\langle A^* \times B^* \rangle\rangle$ **Figure 2.** FSMXML files for label types

For the large range of automata that we want to be able to describe, the need for rational expression is even more striking. For instance, as a consequence of the Kleene-Schützenberger Theorem we may have a transition labeled by a letter whose *weight* is a regular expression.

Rational expressions are well-formed formulas, that naturally correspond to trees and XML is perfectly fitted to describe trees. There is thus not much to say about the translation of an expression into an XML file. Two points have to be noted though.

The expressions we are interested in denote rational series, of course the same as the automata we are considering realize and, for the same reason, the expressions must begin with the description of the type of the semiring of series to which the series they denote belongs. The expressions thus share with automata the tag `<valueType/>` (and this is the reason why we have called it *valueType* and not *labelType*).

As they correspond to weighted automata, our expressions are *weighted expressions* and as the weights may be taken in non commutative semirings, there exist *two* external multiplication operators: a *left* and a *right* one.

It is a major, as well as quite logical, feature of FSMXML that it possesses the possibility of describing rational expressions. As bare letters are also rational expressions, and

with the idea of giving a uniform treatment to the largest class of entities, all labels (of transitions) are represented in FSMXML using the syntax for rational expressions. As an example, Figure 3 shows the description of a transition connecting two states s_0 and s_1 , and labeled by $2(a, b)$.

```
<transition source="s0" target="s1">
  <label>
    <leftExtMul>
      <weight value="2"/>
      <monElmt>
        <monElmt><monGen value = "a"/></monElmt>
        <monElmt><monGen value = "b"/></monElmt>
      </monElmt>
    </leftExtMul>
  </label>
</transition>
```

Figure 3. A transition with input label 'a', output label 'b' and weight '2'

1.3.4. Geometry and Drawing Informations

The tags `<geometricData/>` and `<drawingData/>` are optional child tags of `<automaton/>`, `<state/>`, `<transition/>`, `<initial/>`, and `<final/>`. The former contains coordinates for the automaton and the states, and geometric shapes for transitions. The latter is planned to hold information about the way the automaton and its parts are drawn, but its content is not specified at this stage of our proposal.

2. Design Choices

2.1. Why XML?

While parsing XML is certainly not as efficient as loading a binary file, efficiency is not the first concern when devising an interchange format. The choice of XML simplifies exchanges, manipulations, and future evolutions (adding new tags, attributes, or tokens to support new automata do not invalidate existing files). Frameworks such as DOM [8] or SAX [9] make it easier to build a parser in many languages. An *XML Schema Description* (XSD) document [10] is available on our webpage [6] and many transformations can easily be applied to XML files using languages such as XSLT [11].

2.2. Apparent Verbosity

While XML documents remain human-legible (compared to a binary file at least) this interchange format is meant to be written by computers. We purposely tried to (1) unify the representation of the various automata and (2) refrained from adding any kind of syntactic sugar. In both cases, the intent is to simplify the number of cases an implementation of the format has to deal with.

For instance from the perspective on someone actually typing in an automaton in FSMXML, entering the transition as shown in Figure 3 is cumbersome and one could dream about some kind of syntactic sugar like:

```
<transition source="s0" target="s1" in="a" out="b" weight="2"/>.
```

Our point is that one never writes an XML file representing an automaton by hand (automata are either drawn using a graphical interface, or computed) and from a implementation perspective, the two forms are as easy to input or output. Since the syntax of Figure 3 makes it possible to represent more complex types than the above abbreviation, we have only kept the first: this frees the implementation from having to deal with many special cases. In other words, the verbosity is the result of a simpler grammar, chosen for the sake of simplicity.

2.3. Computable Properties Are Not Part of the Type

Among the ‘structural’ properties of automata, we can distinguish between properties that are *static*, or could be called a *type property*, such as the input alphabet, or the semiring of weights, and properties that we could call *computable* such as ‘being deterministic’, or ‘unambiguous’, or ‘trim’, or ‘functional’ (for a transducer).

As it stands, our proposition can specify static properties but makes no provision for the expression of computable properties. We do agree that such kind of attributes are useful (especially if the format is used for the communication between trusted components). The floor is open for the specification of tokens that will describe these computable properties.

The reason we left these properties aside is that we did not want to organize the format around them. For instance it sounds wrong to specify the type of an automaton by first telling whether it is deterministic or not: this kind of property definitively is not part of the type.

3. Conclusion

The experience gained using an XML format in Vaucanson, with the constraint of being able to define a large variety of automata, has shaped our choices for the proposal on the level both of design and implementation and there have been significant changes since the format we presented at the CIAA 2005 conference [6].

Even though the class of automata initially supported by FSM XML are those targeted by Vaucanson, this format is meant to be extended to encompass the needs of other tools from the community. We believe that the strong typing enforced by the format will give the many communities that use automata the necessary tools to ease such extensions.

References

- [1] D. Raymond and D. Wood. Grail: Engineering automata in C++. <http://www.csd.uwo.ca/Research/grail/>.
- [2] Mehryar Mohri, Fernando C. N. Pereira, and Michael Riley. A rational design for a weighted finite-state transducer library. In Derick Wood and Sheng Yu, editors, *Workshop on Implementing Automata*, volume 1436 of *Lecture Notes in Computer Science*, pages 144–158. Springer, 1997. <http://www.research.att.com/~fsmtools/fsm/>.
- [3] C. Allauzen et al. Openfst: A general and efficient weighted finite-state transducer library. In *Proc. of CIAA'07*, volume 4783 of *LNCS*, pages 11–23, 2007. <http://www.openfst.org>.

- [4] U. Brandes, M. Eiglsperger, and J. Lerne. Graphml - an XML based graph interchange format. <http://graphml.graphdrawing.org>, 2002.
- [5] The Vaucanson Group. Vaucanson, a generic C++ platform for computing automata and transducers (2003–2008). <http://vaucanson.lrde.epita.fr>.
- [6] The Vaucanson Group. Xml proposal for automaton exchanges (2004–2008). <http://vaucanson.lrde.epita.fr/XML>.
- [7] J.E. Hopcroft, R. Motwani, and J.D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 2000.
- [8] W3C. Document object model 2, 2000. <http://www.w3c.org/DOM>.
- [9] D. Megginson. Simple API for XML 2. <http://www.saxproject.org>, 2001.
- [10] XML schema description. <http://www.w3c.org/XML/Schema/>, 2001.
- [11] W3C. XSL transformations. <http://www.w3.org/TR/xslt>, 1999.

A Simple Formalism for Capturing Reduplication in Finite-State Morphology¹

Mans HULDEN^a
and Shannon T. BISCHOFF^b

^a *University of Arizona*

^b *Universidad de Puerto Rico Recinto Universitario de Mayagüez*

Abstract. This paper presents a simple formalism for capturing reduplication phenomena in the morphology and phonology of natural languages. After a brief survey of the facts common in reduplicative elements cross-linguistically, these facts are described in terms of finite-state systems. The principal idea is that an operator can be derived to ensure equivalence of finite discontinuous strings at some level of representation.

Keywords. reduplication, morphology, phonology, finite-state technology

Introduction

Reduplication phenomena are a challenge to finite-state developers of language models for primarily two reasons. Firstly, a constraint that different parts of a string be equal in content is, in the general case, not expressible through finite-state means, and in the specific case, where such duplication is restricted to a finite lexicon, tends to lead to an explosion of the number of states in a finite-state system. Second, in the restricted case, asserting the equality of two parts in a string is not easily expressed through existing finite-state operations.

In this paper we shall give a brief overview of the kinds of reduplication found in natural languages, and, assuming a multi-level approach to morphological analysis through composition, suggest a simple notation to include reduplication phenomena in most grammars—that is, if one is willing to accept the first limitation of the growth of the number of states in an entirely finite-state system that models reduplication.

1. Reduplication

The classical case of reduplication in morphology and phonology is the phenomenon of complete reduplication—a perennial example is that of Bahasa Indonesia (1) or Axin-

¹The authors wish to thank the organizers and participants of FSMNLP 2008, especially Mike Maxwell, for stimulating feedback on an earlier version of this paper. This research was funded in part by a grant provided by Universidad de Puerto Rico Recinto Universitario de Mayagüez.

inca Campa (2), where (typically) pluralization is expressed through reduplication of a complete word:

(1)	Base Form	Reduplication	Gloss
	<i>buku</i>	buku-buku	‘book’ Pl.
	<i>orang</i>	orang-orang	‘man’ Pl. (people)
			[1]
(2)	Base Form	Reduplication	Gloss
	<i>kawosi</i>	kawosi-kawosi	‘bathe’
			[2]

This is a very important type of reduplication, since it appears more or less in every language [3], although not always with an explicit grammatical function as pluralization.²

Another often occurring pattern is the phenomenon where a limited amount of material is copied from a stem that may be longer than the reduplicant (Uw Oygangand):

(3)	Base Form	Reduplication	Gloss
	<i>elmbmben</i>	elbmbelbmben	‘red’
	<i>algal</i>	algalgal	‘straight’
			[5]

A special type of this is the case where only a part is reduplicated, with intervening material (Madurese):

(4)	Base Form	Reduplication	Gloss
	<i>garadus</i>	dusgaradus	‘fast and sloppy’
	<i>abit</i>	bitabit	‘finally’
			[6]

Also, reduplication may not always result in identical material—phonological changes may occur that result in that two (or more) sequences are similar, yet not identical, as in this example from Javanese:

(5)	Base Form	Reduplication	Gloss
	<i>bali</i>	bola-bali	‘return’
	<i>iba</i>	iba-ibu	‘mother’
	<i>udan</i>	udan-udæn	‘rain’
			[7,8]

²English, for instance, apart from the well known *shm*-reduplication (as in *linguistics-shminguistics*), seems to employ the device of total reduplication as a “contrastive focus” method: “I had a JOB-job once. [as opposed to an academic job].” Corpus studies have revealed that this occurs more often than one would expect: see [4] for examples like the one above, or the Corpus of English contrastive focus reduplications at <http://umanitoba.ca/faculties/arts/linguistics/russell/redup-corpus.html>.

The more challenging patterns occur when we find dependencies that cross and produce multiple different partial copies of the base as well as certain affixes. Especially such forms that include phonological change as in Coeur d’Alene:

(6)	Base Form	Reduplication	Gloss
	<i>en’is</i>	<i>ε’en’εn’is</i>	‘little ones went off one by one’
	<i>caq</i>	<i>caqcaqεlipəp</i>	‘he fell on his back’
			[9]

2. Finite-state grammars

The prevalent mode of constructing morphological analyzers for natural languages is that of composing a set of finite-state transducers in sequence, finally producing a transducer that maps strings from an analysis to a surface form, and vice versa. Usually the setup is as follows:³

Lexicon ◦ Rule₁ ◦ ... ◦ Rule_n

This serial mode of creating surface strings from underlying strings lends itself to a particular way of describing reduplication, namely, enforcing the equality of two parts of a string at *some* level of representation. To return to the previous example of the habitual-repetitive form in Javanese and the word *bola-bali*: it would be convenient if a grammar on the lexical level would describe this word more or less as *bali+HabRep*. Then, at a subsequent level, the tag +HabRep would be changed to -bali, giving *bali-bali* (let us disregard for a moment how this could be done), and finally, vowel changes would make the copy different from the base.

Lexicon ◦ Rule₁ ◦ ... ◦ Rule_n
bali+HabRep bola-bali

A rule for Javanese is that if the second vowel of the stem is not a, the first vowel of the left copy changes from a to o and from o to ε, and the second vowel changes to a, i.e. something like:

```
define CRule a → o , o → ε | | .#. C* _ C* [V-a] ?* = „
[V-a] → a | | .#. C* V C* _ ?* = ;
```

Here, C and V represent consonants and vowels, respectively.

Backing up to the operation that produces a copy of the root *bali*: there are methods available for performing this kind of a duplication (e.g. the compile-replace algorithm in the xfst toolkit [10]). However, the apparent simplicity of reduplication is somewhat muddled by the complexity of current methods to handle the problem. Working with a serialist description, one where regular relations are composed sequentially, our survey shows that almost every type of reduplication could be handled if there were some simple notational way to express a constraint about the similarity of strings at some level of representation. In the example above, this kind of a constraint would have to be enforced before composition with CRule.

³We assume an xfst-like notation here following [10], since that is the tool we used for our testing.

3. Enforcing equality

We have experimented with defining an operator $\text{EQ}(L, \text{left}, \text{right})$ that takes three regular languages as arguments, L (the language of interest) and left and right (two arbitrary delimiters), such that it filters from L the set of strings where everything delimited by $\text{left}, \text{right}$ is not equal. If such an operator existed, the above example of *bola-bali* could be handled as follows: we could, in the lexical level, define the roots in such a way that they are surrounded by special symbols (delimiters), which we here call $<$ and $>$. For the sake of generality, $\text{left}, \text{right}$ could be arbitrary regular languages, however, single-symbol languages probably suffice for grammar writing. With this in mind we return to our Javanese example *bola-bali*, illustrating the implementation of EQ .

<u>Lexicon</u>	Rule ₁	Rule ₂	Rule ₃
bali+HabRep	$\rightarrow \text{bali}<\Sigma^*>$	$\rightarrow <\text{bali}><\Sigma^*>$	$\rightarrow <\text{bali}><\text{bali}>$

- Rule₁: $\text{+HabRep} \rightarrow <\Sigma^*>$ ($\{<, >\} \notin \Sigma$)
- Rule₂: surround reduplicant with $<, >$
- Rule₃: enforce equality of all substrings that are inside $<, > — \text{EQ}(<, >)$ applies here.

Rule ₄	Rule ₅
$<\text{bali}><\text{bali}>$	$\rightarrow <\text{bola}><\text{bali}> \rightarrow \text{bola-bali}$

- Rule₄: $\text{a} \rightarrow \text{o} \dots$
- Rule₅: remove brackets, etc.

For the purposes of testing the notation, we have approximated EQ as follows.

1. Set n to 2.
2. Extract only the strings from L that contain exactly n pairs of $\text{left}, \text{right}$. Then create $L_1 \dots L_n$ from the strings between left and right . If this yields the empty language, go to 6.
3. Intersect $L_1 \cap \dots \cap L_n$, yielding L
4. Discard any resulting arcs in L that induce a cyclic path
5. Extract a word list from L , add to list W . Increase n , go to 2.
6. Create from W the language $L \ \& \ \text{CO}(w_1) \mid \dots \mid \text{CO}(w_n) \mid \sim \$ [L \ ?^* \ R \ ?^* \ L \ ?^* \ R]$ where $\text{CO}(X)$ is $[\sim \$ [L \mid R] \ L \ X \ R \ \sim \$ [L \mid R]]^*]$
7. create $\text{EQ}(L, \text{left}, \text{right}) = L \ \& \ W'$

This is not meant to be an actual practical algorithm, neither efficient nor maximally general (given finite-state limitations), but rather a first approximation to practically test the simplicity of reduplicating grammars, given the availability of an operation EQ , or something equivalent.

Where the EQ function becomes most useful and transparent is in describing the equality of discontinuous parts of strings, as in the Madurese plural example *duşgaradus*, where the final syllable of the root *garadus* is prefixed. With the EQ function, one can, on the lexical level, generate the roots, change +Pl tags into a prefixed $<\Sigma^*>$ sequence, surround the final syllable of the root with brackets $<$ and $>$, and compose the relation with $\text{EQ}(L, <, >)$, where L is the range (lower projection) of the transducers composed so far, giving a sequence of derivations as follows:

```

garadus+Pl
< ... >garadus
< ... >gara<dus>
<dus>gara<dus>      (EQ(L,<,>) applies here)
dusgaradus

```

As above, we will want to have a rule that removes our special symbols after EQ is no longer needed.

As a another example, we also implemented the reduplication pattern of Warlpiri, of which some examples are given here:

(7) Base Form	Reduplication	Gloss
<i>pakarni</i>	<i>pakapakarni</i>	‘hit (?)’
<i>wantimi</i>	<i>wantiwantimi</i>	‘fall’
<i>tiirlparnkaja</i>	<i>tiitiirlparnkaja</i>	‘split lengthwise’
<i>pangurnu</i>	<i>pangupangurnu</i>	‘dig’
[11]		

The reduplication pattern works as follows: an instance of $C \vee (C) (C) \vee$ is copied from the stem, and prefixed. Starting with the lexical level, that contains strings such as: Redup+pakarni, we change +Redup to a prefixed string $\langle \Sigma^* \rangle$, yielding, in this example: $\langle \Sigma^* \rangle$ pakarni. We then compose this level with the rule:

```
define MarkRedup [C V (C) (C) V] -> %< ... %> || %> _;
```

marking an instance of the prosodic pattern found in the root. Following this, we apply $EQ(L, <, >)$, and remove the bracket symbols, yielding derivations such as:

```

pakarni+Redup
< ... >pakarni
< ... ><paka>rni      (EQ(<,>) applies here)
<paka><paka>rni
pakapakarni

```

For more complicated patterns, such as the crossing partial reduplications of Coeur d’Alene in example (6), we need to resort to several different kinds of brackets, enforcing EQ on multiple occasions. Even so, the notation is transparent and yields grammars that are quite clear, as the following simplified example illustrates (recall that both the base *caq* and the morpheme *-ip* are reduplicated). In the following the plus “+” is employed throughout for clarity to indicate a morpheme boundary, in implementation this plus would be removed by a rule akin to the rule that removes bracketing.

Lexicon Rule₁
 $caq+Aug+Prog+ip+AfRedup \rightarrow caq+\langle \Sigma^* \rangle + [\Sigma^*] + ip + (\Sigma^*)$

- Rule₁: +Aug $\rightarrow \langle \Sigma^* \rangle$, +Prog $\rightarrow [\Sigma^*]$, +AfRedup $\rightarrow (\Sigma^*)$

Rule₂

$$caq+\langle \Sigma^* \rangle + [\Sigma^*] + ip + (\Sigma^*) \rightarrow <c[caq]> + \langle \Sigma^* \rangle + [\Sigma^*] + (ip) + (\Sigma^*)$$

- Rule₂: surround reduplicants with $<, >, [,], (,)$

$$\text{Rule}_3 \quad <c[aq]> + <\Sigma^*> + [\Sigma^*] + (ip) + (\Sigma^*) \rightarrow <c[aq]> + <c[aq]> + [\Sigma^*] + (ip) + (\Sigma^*)$$

$$\text{Rule}_4 \quad <c[aq]> + <c[aq]> + [\Sigma^*] + (ip) + (\Sigma^*) \rightarrow <c[aq]> + <c[aq]> + [aq] + (ip) + (\Sigma^*)$$

$$\text{Rule}_5 \quad <c[aq]> + <c[aq]> + [aq] + (ip) + (\Sigma^*) \rightarrow <c[aq]> + <c[aq]> + [aq] + (ip) + (ip)$$

- Rule₃, Rule₄, Rule₅: enforce equality of all substrings that are inside brackets — *EQ applies here (elements of brackets will be reduplicated as well, but will be removed later).*

$$\text{Rule}_6 \quad <c[aq]> + <c[aq]> + [aq] + (ip) + (ip) \rightarrow <c[aq]> + <c[aq]> + [aq] + (ip) + (\emptyset p)$$

- Rule₆: $i \rightarrow \emptyset \dots$

$$\text{Rule}_7 \quad <c[aq]> + <c[aq]> + [aq] + (ip) + (\emptyset p) \rightarrow caqcaqaipep$$

- Rule₇: remove brackets, etc.

As can be seen, the use of bracketing makes the process not only rather transparent, but also rather simple even in a rather complex example such as *caqcaqaipep* with multiple reduplicative forms and phonological change.

The examples thus far illustrate rather nicely how our proposed formalism can be applied to facts of reduplication within finite-state methods. There is however a curious type of reduplication found in Bambara that would pose challenges to finite-state methods, and thus our formalism. In Bambara the compounding of nouns, in theory, can be *infinite*. Further, once compounded, these noun forms can be reduplicated as in the following example taken from [8]. First we present the nouns and their glosses, the compound and its gloss, next we show the compound reduplicated. It should be noted that an *o* appears between the base and the reduplicant once reduplication has occurred.

(8) Base Form	Gloss
<i>malo</i>	<i>rice</i>
<i>nyinina</i>	<i>searcher</i>
<i>filèla</i>	<i>watcher</i>
Compound Form	Gloss
<i>malo-nyinina-filèla</i>	<i>rice-searcher-watcher</i>
Reduplication	Gloss
<i>malo-nyinina-filèla o malo-nyinina-filèla</i>	<i>whichever rice-searcher-watcher</i>
	[8]

It can easily be imagined how the above example could be captured in the proposed formalism.

$$\underbrace{\text{Lexicon}}_{\text{malo-nyinina-filèla+Redup}} \quad \circ \text{Rule}_1 \circ \dots \circ \text{Rule}_n \quad \text{malo-nyinina-filèla} \circ \text{malo-nyinina-filèla}$$

Naturally, the +Redup would be changed at some level to $\circ <\Sigma^*>$. Of course, the infinite recursion of the noun elements in the base form would make this impossible using finite-state technology alone. However, other computational means could be employed to tackle this particular, and extremely rare, form of reduplication. Thus, exceptions such as this do not affect the motivation for the proposed formalism.

For clarity we offer one final example of how the proposed formalism might be implemented when phonological change is involved. In Malay, nasalization of vowels and bilabial glides is triggered in the base form by reduplication. The following examples illustrate the phenomenon.⁴

(9) Base Form		Reduplication	
<i>hamǎ</i>	‘germ’	<i>hǎmǎ-hǎmǎ</i>	‘germs’
<i>waŋĩ</i>	‘fragrant’	<i>ŵǎŋĩ-ŵǎŋĩ</i>	(intensified)
<i>aŋǎn</i>	‘reverie’	<i>ǎŋǎn-ǎŋǎn</i>	‘ambition’
<i>aŋĕn</i>	‘wind’	<i>ǎŋĕn-ǎŋĕn</i>	‘unconfirmed news’

[12]

Taking *waŋĩ* as our example we can see how the delimiters used to ensure equality, in this case the brackets $<$ and $>$, can be used to ensure the relevant phonological change via a series of simple re-write rules before EQ is applied. To begin, we want to define all the elements that are nasalized and thus trigger nasalization in reduplication (here we simply use those from our limited data set for explication).

define N [*ǎ* | *ĩ* | *ŵ* | *ĩ* | *ĕ*] ;

Next, we implement the grammar as we have done above. However we make three changes: we place the rule for bracketing the base before the phonological rules relevant to nasalization. Then we add the rule for changing the relevant morpheme to $<\Sigma^*>$. Next we add the rule that implements EQ.

Lexicon Rule₁
waŋĩ+Redup → $<waŋĩ>+Redup$

- Rule₁: surround reduplicant with $<, >$ — *Note this is different from above.*

Rule₂
 $<waŋĩ>+Redup \rightarrow <\tilde{w}\tilde{a}\tilde{\eta}\tilde{\imath}>+Redup$

- Rule₂: $a \rightarrow \tilde{a} \mid \mid < ?^* N ?^* _ , _ ?^* N ?^* > , w \rightarrow \tilde{w} \mid \mid < ?^* N ?^* _ , _ ?^* N ?^* > \dots$
 (Changes relevant elements in brackets to nasalized forms.)

Rule₃
 $<\tilde{w}\tilde{a}\tilde{\eta}\tilde{\imath}>+Redup \rightarrow <\tilde{w}\tilde{a}\tilde{\eta}\tilde{\imath}>+<\Sigma^*>$

- Rule₃: Redup → $<\Sigma^*>$

Rule₄
 $<\tilde{w}\tilde{a}\tilde{\eta}\tilde{\imath}>+<\Sigma^*> \rightarrow <\tilde{w}\tilde{a}\tilde{\eta}\tilde{\imath}>+<\tilde{w}\tilde{a}\tilde{\eta}\tilde{\imath}>$

- Rule₄: enforce equality of all substrings that are inside $<, >$ — *EQ(L, <, >) applies here.*

⁴Thanks to Mike Maxwell for bringing this example to our attention.

$$\begin{array}{c} \text{Rule}_5 \\ \langle \tilde{w}\tilde{a}\tilde{n}\tilde{j}\tilde{i} \rangle + \langle \tilde{w}a\tilde{n}\tilde{j}\tilde{i} \rangle \rightarrow \tilde{w}\tilde{a}\tilde{n}\tilde{j}\tilde{w}\tilde{a}\tilde{n}\tilde{j}\tilde{i} \end{array}$$

- Rule₅: remove brackets, etc.

Once again, in this way we arrive at the relevant reduplicated form.

4. Conclusion

There are many advanced methods available for building finite-state networks that encode languages that feature non-concatenative phenomena, including reduplication: e.g. the compile-replace technique [13], adding extra memory to the parsing algorithm [14], or other techniques where reduplication is semantically encoded into the automata [15]. However, for most such phenomena—including perhaps even simple vowel-lengthening—a very compact notation would allow the developer to, at some level of representation, assert that discontinuous parts of a string are equal. Such a function can either be compiled directly into automata, assuming the reduplicants are finite, or, if one is concerned about the size of the transducers, used as the basis for a run-time constraint where the transducer is split into two parts: pre-equality, and post-equality, and equality enforcement happens on the fly as the two are “virtually composed.”

References

- [1] R.R. Macdonald and S. Darjowidjojo. *A Student's Reference Grammar of Modern Formal Indonesian*. Georgetown University Press, Washington, DC, 1967.
- [2] D.L. Payne. *The phonology and morphology of Axininca Campa*. University of Texas at Arlington, Arlington, TX, 1981.
- [3] E. Moravcsik. Reduplicative constructions. In J. Greenberg, editor, *Universals of Human Language*, volume 3, pages 297–334. Stanford University Press, Stanford, CA, 1978.
- [4] J. Ghomeshi, R. Jackendoff, N. Rosen, and K. Russell. Contrastive focus reduplication in English (the Salad-Salad paper). *Natural Language & Linguistic Theory*, 22(2):307–357, 2004.
- [5] B.A. Sommer. The shape of Kunjen syllables. In D.L. Goyvaerts, editor, *Phonology in the 1980s*. Story-Scientia, Ghent, 1981.
- [6] A. Stevens. *Madurese phonology and morphology*. American Oriental Society, New Haven, CT, 1968.
- [7] P. Kiparsky. The phonology of reduplication. manuscript, Stanford University, 1987.
- [8] R. Sproat. *Computational Morphology*. MIT Press, Cambridge, MA, 1992.
- [9] G. Reichard. Coeur d'Alene. In F. Boas, editor, *Handbook of American Indian Languages*, volume 3, pages 515–707. J. J. Augustin, New York, 1938.
- [10] K. Beesley and L. Karttunen. *Finite-State Morphology*. CSLI, Stanford, 2003.
- [11] D.G. Nash. *Topics in Warlpiri Grammar*. PhD thesis, MIT, 1980.
- [12] R. Kager. *Optimality Theory*. Cambridge University Press, Cambridge, 1999.
- [13] K. Beesley and L. Karttunen. Finite-state non-concatenative morphotactics. In *Proceedings of the 38th Annual Meeting of Association for Computational Linguistics*, pages 191–198, 2000.
- [14] Y. Cohen-Sygal and S. Winter. Finite-state registered automata for non-concatenative morphology. *Computational Linguistics*, 32(1):49–82, 2006.
- [15] M. Walther. Finite-state reduplication in one-level prosodic morphology. In *Proceedings of the first conference on North American chapter of the Association for Computational Linguistics*, pages 296–302, 2000.

Applying Finite State Morphology to Conversion Between Roman and Perso-Arabic Writing Systems

Jalal MALEKI¹, Maziar YAESOUBI and Lars AHRENBORG

*Department of Computer and Information Science,
Linköping University, SE-581 83 Linköping, Sweden*

jma@ida.liu.se, maziar.yaesoubi@gmail.com, lah@ida.liu.se

Abstract. This paper presents a method for converting back and forth between the Perso-Arabic and a romanized writing system for Persian. Given a word in one writing system, we use finite state transducers to generate morphological analysis for the word that is subsequently used to regenerate the orthography of the word in the other writing system. The system has been implemented in XFST and LEXC.

Keywords. Transliteration, Finite State Transducers, Perso-Arabic Script, Romanization

Introduction

We present a method for converting between two scripts for Persian: the traditional Perso-Arabic writing system [1][2] and a romanized script called Dabire [3]. The conversion system, which is being developed using Xerox LEXC and XFST tools [4], uses finite state transducers for modeling analysis and generation of word forms, phonological alternations and orthographical conventions. Although our implementation is specific to Persian spoken in Iran, the orthographical conversion model is general and can be applied to any language with multiple scripts.

The essence of our approach is as follows. Let M_1 and M_2 denote morphological analysis transducers for two possible scripts of a language and let L denote a transducer that implements a stem lexicon mapping stems from one script to the other. Ideally, we can construct a script conversion transducer by composing these transducers as thus: $M_1^i \otimes L \otimes M_2$. Here M_1^i denotes the inverse of M_1 and \otimes is the operation for transducer composition.

Persian (an Indo-European language) is mainly written in variations of the Perso-Arabic script (PA-Script) [2][5]. The Latin script was officially used in Tajikistan in the early days of the Soviet republic but was quickly abandoned in favor of the Cyrillic script [6]. Nowadays, however, the Latin script is used extensively in text-based mobile and electronic communication among Persian speakers.

¹We would like to thank Prof. Klaus Lagally and Mr. Ola Leifler for their generous help in resolving some of the typesetting issues of this paper.

Table 1. Mapping /e/ to PA-Script Graphemes. اوپئه is the transcription of the Hawaiian word 'Opae (shrimp).

/e/	Word Initial	Segment Initial	Segment Medial	Segment Final	Isolated
V, VC, VCC	ا ابراهيم	و سوئز	ئ مطمئن	ه اوپئه	ه نيكاراگوئه
CVC, CVCC		ـ آيستن	ـ بکش	ـ کدير	
CV		ـ خواهش	ـ بکشيد	ه پروانه	ه آتشکده

The extensive amount of information published on the Internet in PA-Script and varieties of proposed Latin-based scripts motivates our work in bridging the gap by trying to understand the relationship between these scripts and also automatically converting between them. Our aim is to create a platform for applications such as multi-script chat, search, data mining, and indexing for libraries.

1. Persian Script Conversion Challenges

In this section we will present a brief description of the PA-Script and list a number of challenging problems in using morphological anlysis for script conversion. We are interested in converting between two different writing systems for Persian. First, the traditional PA-Script used in Iran, which is an extension of the Arabic script and includes some Persian-specific graphemes as well as some minor revisions to the orthographic rules of the Arabic script. The second writing system we use in our implementation is a Latin-based phonemic transcription called Dabire that is described in [3]. Since the correspondence between Persian phonemes and graphemes of Dabire is straightforward and the conventions of the script are similar to other Latin-based scripts, we will not discuss it in any detail. We will, however, give a short description of the traditional PA-Script below and when necessary mention specifics of Dabire.

PA-Script is a semi-cursive writing system in which words are written from right to left by joining the appropriate graphemes. The typed variations of the writing system simulate the hand-written semi-cursive style and inherit its properties. The correspondence between consonants and graphemes representing them is relatively straightforward, whereas vowel representation is more complicated. Table-1 shows how the short vowel /e/, for example, may be represented in various contexts. The diacritics َ, ِ, ُ can be used to indicate the presence of a short vowel (e, a, o respectively), ْ is used to indicate absence of a vowel and ّ is used to indicate gemination. However, these diacritics are usually not used unless there is a pedagogical reason for including them. Here is an expression with three words: يك سيب قرمز (a red apple) which in the fully vocalized version would be written as يَک سَيِّبِ قَرْمَزْ (yek sib e qermez).

An alphabetic word is written as a sequence of one or more segments written from right to left. Segments are separated by a zero-width space. In this paper, we use the word

segment to refer to a sequence of conjoined graphemes. A segment is a orthographical notion and does not necessarily coincide with a phonological or morphological unit.

The cursive nature of the writing system necessitates multiple allographs for a grapheme. An allograph is essentially the adaptation of a grapheme so that it can properly join its neighboring allographs. There are four different positions in which a cursive allograph can appear: *Segment-Initial*, *Segment-Medial*, *Segment-Final* and *Isolated*. Some graphemes *fully-cursive* and have four allographs, one for each position. Others only join their predecessor graphemes and do not join their successors and this essentially means that the grapheme either appears on its own or it ends a segment - we call these *semi-cursive* allographs. The graphemes ا, د, ذ, ر, ز, ژ and و are semi-cursive and never join the following grapheme and therefore terminate the segment in which they appear.

The rest of this section discusses some of the problematic issues related to conversion of scripts.

1.1. Analysis Problems: PA-Script

One major problem in the analysis process of real world texts is related to tokenization. Megerdumian [7] gives a fair account of tokenization problems in processing Persian text and suggests some remedies. An important cause of tokenization problems is that word boundaries are not always marked correctly. Words ending in semi-cursive graphemes are not delimited properly, for example, the sentence "کار را کرد و رفت" (Did the work and left) may be written as "کار را کرد و رفت" without any spaces between the five words constituting it. The reason for the latter being readable at all is that all words end with semi-cursive graphemes that do not join to their successors. This is usually not a problem for the human eye familiar with the script, but to an automatic tokenizer the latter form would appear as a single token. Another example is when constituents of a complex token are separated with a normal space rather than with the zero-width non-joining space (ZWNJ) [8] which is the correct delimiter for separating orthographical segments. For example, پروانه (butterfly) and وار (like) can be joined to form the compound word پروانه وار (like a butterfly) where the two words are correctly separated using a ZWNJ character. However, a less carefully typed version (پروانه وار) may use space rather than a ZWNJ as delimiter creating two separate tokens.

Another issue in analysis is that Persian verbs have two stems: *present stem* and *past stem*. The present stem can in principle be derived from the past stem, see [9] for an implementation of this derivation process. However, since the number of verbs is limited, one can represent the present and the past stems separately as in [7]. Yet another compli-

cation in the analysis process is the existence of the so called long-distance dependencies in verbs [7].

1.2. Generation Problems: PA-Script

Given the morphological information about a word (a stem and a set of feature tags), some of the main problems in generating a PA-Script word involve vowel representation, representation of phonological alternations and also generation of ZWNJ space in compound words.

PA-Script has a relatively ad hoc set of conventions for writing compound words which allows a large number of exceptions. These are listed in a recent publication by the Persian Academy [1]. Some authors dispute the adequacy and the accuracy of these conventions [10]. However, the general principle is to write compounds in a semi-open format² to make sure that the graphic identities of the sub-words of a compound are preserved as much as possible in order to minimize ambiguities. In computer-based texts, a ZWNJ-space is used to separate the constituents of a compound in order to override the cursive nature of the orthography. In contrast to PA-Script, Dabire has a simple set of conventions [3] for writing compounds that clearly indicate when words should be written in open or closed format. In short, just like some European languages such as Swedish, the default format for writing compound words in Dabire is the closed format, whereas, the preferred format in PA-Script is the semi-open format.

In PA-Script, some graphemes have multiple roles, for example, **و** (with the allo-graphs **و**, **و**, **و** and **و**) is used for denoting /h/ as well as word final /a/ and /e/. Here are some examples:

[*kuh*, **کوه**, **kwh**, *kuh*, mountain]

[*kuce*, **کوچه**, **kwch**, *kutʃe*, alley]

[*na*, **نه**, **nh**, *næ*, no]

When such a word forms the non-final sub-word of a compound token, the **و** being fully-cursive can join the initial grapheme of the next sub-word and its shape will change from the segment-final or isolated form (**و**, **و**) to the segment-initial or segment-medial (**و**, **و**). Since **و** represents a vowel only if it occurs in the segment-final or isolated form, changes in its shape may create ambiguities for the reader. It is therefore fine to write the plural of **کوه** (*kuh*) as **کوهها** (*kuhhâ*) but it is not good practice to write the plural of **کوچه** (*kuce*) as **کوچهها** (*kucehâ*). It should be written as **کوچه‌ها**. Similarly, **کوچه‌ای** (an alley) is clearly a better choice compared to **کوچه‌های**.

²We call this format *semi-open* to distinguish it from the open format in English that uses space to separate parts of the compound [11].

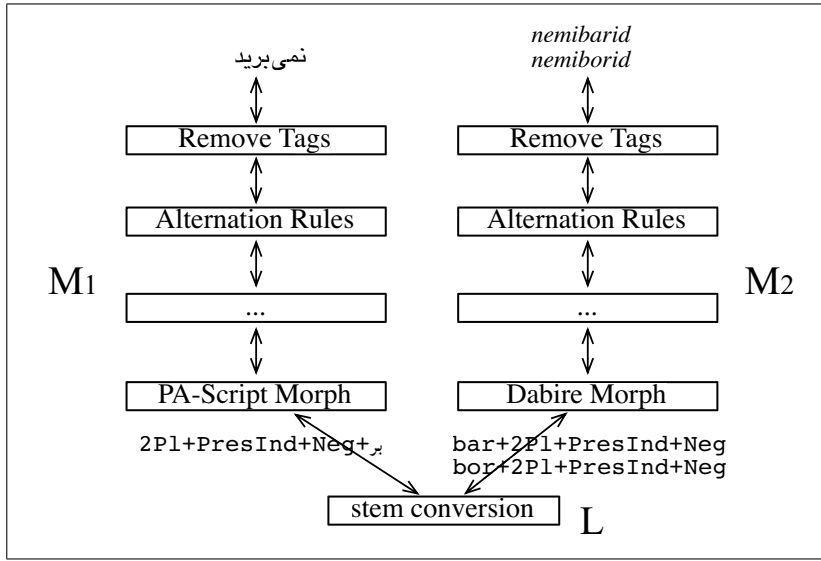


Figure 1. The composition of transducers that use finite state morphological analysis and a stem transcription lexicon to transcribe words of Dabire to PA-Script and vice versa.

2. The Implementation

Our system is implemented using Xerox LEXC and XFST [4] and currently consists of: a LEXC-module for specifying morphology for Dabire, a similar LEXC-module for PA-Script,³ a syllabification method implemented in XFST (see [12]), a simple transducer that implements a lexicon for stems in Dabire and PA-Script and finally the main XFST-module that integrates the whole system and contains miscellaneous transducers such as alternation rules and ZWNJ-space insertion rules.

Although finite state transducers are bidirectional, we have designed our transducers as generators, that is, we define how words can be constructed by systematically attaching morphological features to word stems. The complete finite state transducer for converting from Dabire to PA-Script is defined as a multi-level composition of transducers as shown in Figure-1. The left part of the figure implements the PA-Script morphology transducer (M_1), and the right part the Dabire morphology transducer (M_2). The box at the bottom of the diagram is a simple transducer (L) that maps between stem transcriptions.

As we mentioned in the introduction, an orthographical conversion system can be subsequently defined as the composition of these transducers as follows: $M_2^i \otimes L \otimes M_1$. Here M_2^i denotes the inverse of M_2 and \otimes denotes transducer composition operation (\circ in XFST).

The following example trace shows how the verb نمی برید (take+2Pl+PresInd+Neg)

³The LEXC-modules for the two scripts are very similar and it is possible to generate one from the other using the lexicon transducer, but we have not exploited this possibility yet.

is analyzed to the present stem بر of the verb بردن (to take). In the trace, نمی‌برید and بر are shown as "nmybryd" and "br" respectively. The steps in the trace, numbered as M1* and M2*, illustrate various stages in the analysis process in M_1 and M_2 transducers.

```
M11. br+2Pl+PresInd+Neg
M12. br+2Pl+PresInd^Dur+Neg
M13. [br]+2Pl+PresInd^Dur+Neg
M14. nmy[br]+2Pl+PresInd^Dur+Neg
M15. nmy[br]yd+PresInd^Dur+Neg
M16. nmybryd
```

Dabire-morphology produces similar trace,

```
M21. bar+2Pl+PresInd+Neg
M22. bar+2Pl+PresInd^Dur+Neg
M23. [bar]+2Pl+PresInd^Dur+Neg
M24. nemi[bar]+2Pl+PresInd^Dur+Neg
M25. nemi[bar]id+PresInd^Dur+Neg
M26. nemibarid
```

It is clear from these examples that inverting either M_1 or M_2 together with a transducer for stem conversion (stem dictionary) enables us to construct a FST for converting from one writing system to another.

Finally, the implementation constitutes a relatively large number of transducers that implement the rules and conventions of the writing system. For example, the rule $i \rightarrow [a \ y] \ || \ . \# \ . _$ would replace word-initial occurrences of i with $a \ y$ which at a later stage is transliterated to ای (see Table-1). This particular rule covers one instance of the orthography of i which occurs in syllables of the form V, VC, VCC and would be written independent of other segments (for example, in کارخانه‌ای (a factory)).

Finally, as an example illustrating peculiarities of Persian orthography in our XFST implementation we include part of the rules for inserting zero-width spaces in the context of compound words in PA-Script. In the following rules, ZWNJ is shown as +Z:

```
define paZWJN [
  [..] -> %+Z ||
    %+Pre [[? - CmpndTag]* - [ b h | b y | h m ]]
    _ [CmpndTag - %+Pre]
  .o.
  [..] -> %+Z || %+Num [[? - CmpndTag]* _ [CmpndTag]
  .o.
```

```
[...] -> %+Z || b _ CmpndTag b
...]
```

The first rule states the convention that PA-Script prefixes other than به, بی and هم (shown as bh, by and hm in the rule) should not join the rest of the word [1]. The second part implements another orthographic convention of PA-Script that suggests that numbers initiating a compound word, should be separated from the rest of the word using a ZWNJ-space, for example,

[*panjzel'i*, پنج ضلعی, **pnj-zl'y**, *pændʒzelʔɪ*, pentagon]

is a compound built using [*panj*, پنج, **pnj**, *pændʒ*, five] and [*zel'i*, ضلعی, **zl'y**, *zelʔɪ*, sided].

Finally, the third rule which is the first instance of a series of replace rules (one for each consonant) indicates that if one constituent of a compound ends with the same grapheme that initiates the following sub-word, then the graphemes should be separated by a zero-width space. For example, نیک (good) which ends with a "k" and کردار (deed) which starts with a "k" can join to form a compound that can either be written as نیککردار or نیک کردار. However, the latter is preferred since it discourages the reader from inserting a vowel after the first word. Essentially, this sort of complications is the price the PA-Script has to pay for continuing to avoid short vowel representation.

3. Evaluation

Our system has not been evaluated in a real setting mainly because the stem lexicon is small (500 words), the morphological rule-base does not cover all paradigms and some of the necessary orthographical conventions are not represented. In particular, the rule-base for handling affixes, adverbs and complex verbs is yet to be completed. However, the results of the following evaluation are encouraging.

In a limited evaluation experiment, we randomly selected 448 words from Tehran University Bijankhan Corpus [13] (which uses PA-Script), added all necessary stems to the stem lexicon and tested the system in the conversion direction from PA-Script to Dabire.

The transcriptions produced by the system were divided into three groups: *correct*, *failure* and *partially-correct*. An answer was classified as correct when the list of the generated transcriptions for the input word only included correct answers. For example, نبرد has a number of alternative analyses and can, therefore, be correctly romanized as

nabard (fight+Noun+Sg), *nabord* (take+Past+3Sg), *nabarad* (take+Pres+3Sg) or *naborad* (cut+Pres+3Sg). An answer was classified as partially-correct if it contained at least one orthographically inaccurate transcription or a non-word (over-generation). Finally, in those cases where no transcriptions were produced, the answer was classified as failure.

The system produced 88% correct transcriptions (394 cases), failed in 4% of the cases (17 words) and generated partially-correct answers for the remaining 8% (37 words). Every partially-correct answer contained at least one correct transcription and the total number of generated transcriptions for the 37 words amounted to 162 of which 54% (88 transcriptions) were correct and 46% (74) were either non-words or orthographically inaccurate.

Failures were either due to the incompleteness of the morphological rules or existence of morphologically rare compound words - that is words whose formations do not follow general morphological rules.

4. Conclusion

In this paper, we have briefly described a general approach to the problem of automatic conversion between two alternative scripts of a language. The main idea presented here is to generate morphological analysis for a word written in one writing system and then use the analysis to produce the orthography for the word in the other writing system. The case of romanized and Perso-Arabic writing systems for Persian is specially interesting since the writing systems are very different and enjoy different writing conventions.

The core of our implementation consists of finite state transducers for representing morphological analysis and production, phonological alternations and orthographical conventions of the scripts. The system is implemented using Xerox LEXC and XFST tools [4] [14].

Although FSM-technology has been extensively used in many applications, our use of the technology for automatic transcription between multiple scripts for Persian is unique. Related work includes [15] and [16] that apply XFST-technology to Arabic transcription and transliteration. [7] applies XFST to morphological analysis of Persian and [17] successfully uses the technology for constructing a pronunciation dictionary for Turkish. Unfortunately, we have not been able to build on earlier systems for Persian since their software is proprietary.

Our future work involves extending the system to cover all morphological paradigms and a large stem lexicon. Furthermore, we intend to extend the system so that it can convert words that are not represented in the lexicon. In our earlier work [12] we have used a syllabification-based approach for converting correct Dabire-words that lack lexical representation. We are also working on a system that uses HMM-techniques similar to [18] for transcription of PA-Script words which lack lexical representation.

References

- [1] *Dastur e Khatt e Farsi (Persian Orthography)*. Tehran, 2003.
- [2] S. Neysari. *A Study on Persian Orthography - (in Persian)*. Sâzmân e Câp o Enteshârât, 1996.
- [3] J. Maleki. A Romanized Transcription for Persian. In *Proceedings of Natural Language Processing Track (INFOS2008)*, Cairo, 2008.

- [4] K.R. Beesley and L. Karttunen. *Finite State Morphology*. CSLI Publications, 2003.
- [5] M.S. Adib-Soltâni. *An Introduction to Persian Orthography - (in Persian)*. Amir Kabir Publishing House, Tehrân, 2000.
- [6] J.P. Perry. A Tajik Persian Reference Grammar. In *Handbook of Oriental Studies*. Brill, Leiden, 2005.
- [7] K. Megerdooimian. Finite-State Morphological Analysis of Persian. In A. Farghaly and K. Megerdooimian, editors, *Proceedings of the Workshop on Computational Approaches to Arabic Script-based Languages*, pages 35–41, 2004.
- [8] B. Esfahbod. Persian Computing with Unicode. In *25th Internationalization and Unicode Conference*, Washington, DC, 2004.
- [9] R. Ziai. Finite State Methods Applied to Verbal Inflection in Persian. Master's thesis, Eberhard-Karls Universität, Tübingen, 2006.
- [10] R.R.Z. Malek. *Qavâed e Emlâ ye Fârsi*. Golâb, 2001.
- [11] R.M. Ritter. *The Oxford Guide to Style*. Oxford University Press, 2002.
- [12] J. Maleki and L. Ahrenberg. Converting Romanized Persian to Arabic Writing System. In *Proceedings of the LREC2008, Marrakech*, 2008.
- [13] M. Bijankhan. Bijankhan Corpus. Tehran University, <http://ece.ut.ac.ir/dbrg/bijankhan/>, 2008.
- [14] R.M. Kaplan and M. Kay. Regular models of phonological rules systems. *Computational Linguistics*, 20(3):331–378, 1994.
- [15] K.R. Beesley. Romanization, transcription and transliteration. Xerox. <http://www.xrce.xerox.com/competencies/content-analysis/arabic/info/romanization.html>, 1996.
- [16] K.R. Beesley. Arabic finite-state morphological analysis and generation. In *Proceedings of COLING'96*, Copenhagen, 1996.
- [17] K. Oflazer and S. Inkelas. The architecture and the implementation of a finite state pronunciation lexicon for Turkish. *Computer Speech and Language*, 20:80–106, 2006.
- [18] Y. Gal. An HMM approach to vowel restoration in Arabic and Hebrew. In *Proceedings of the ACL-02 workshop on computational approaches to semitic languages*, pages 1–7, 2002.

Morphisto - An Open Source Morphological Analyzer for German

ANDREA ZIELINSKI ^a, CHRISTIAN SIMON ^a

^a *Institute for German Language,
 R 5, 6 - 13, 68161 Mannheim, Germany
 E-mail: {zielinski, simon}@ids-mannheim.de*

Abstract. This paper presents the development of an open-source morphology tool for German integrated into a grid-based environment. Departing from the SFST-based SMOR tools (Schmid et al. [1]), we have implemented a *minimal* lexicon component that works in tandem with the morphological tool. Tests on a list of 30,000 high-frequency German words show that the recognition rate is comparable to other systems with even larger lexicons. Additional tools for the management of lexical data and services built on top of the finite-state transducer are also integrated as web services in the grid, so that all resources can be shared easily among lexicographers, linguists, and finite-state developers.

Keywords. Computational German Morphology, Finite-State Transducer, Grid, SFST

Introduction

As part of the TextGrid project [2], we have developed an open-source morphological tool for the German language. Although numerous resources exist (i.e., WMTrans, GerT-WOL, INXIGHT, Morphix, TAGH, etc.), no out-of-the-box broad coverage morphology system is freely available at present. This is especially true for the lexicon, which is an essential component of any morphology system for sophisticated real-life applications. As is well known, finite-state transducers are the first choice in computational morphology, because they are declarative, can be used bidirectionally for analysis and generation, and are efficient in time and space. Thus, we based our work on the finite-state toolkit SFST (Schmid [3]) that comes with the morphological grammar SMOR (Schmid [1]) and a small lexicon with approximately 1,000 example entries. As a consequence, we adopted the underlying model-theoretic assumptions in SMOR and encoded the required lexical information according to that framework. Our aim was to cover the 30,000 most frequent words in the German language: The DeReWo list¹ of lemmas compiled from the DeReKo corpus² provided by the IDS Mannheim has been chosen as the training set.

¹<http://www.ids-mannheim.de/kl/derewo/> DeReWo List and User Documentation (C) IDS, Mannheim, 2007

²<http://www.ids-mannheim.de/kl/projekte/korpora/>

As the project focuses on supporting linguists and philologists working on historical language resources, we plan to extend the morphological analyzer so that it covers other diachronic stages of German as well. At present, we are concentrating on contemporary German and the early stages of New High German (NHD).

1. Starting Point: SMOR & SFST

The basis of Morphisto is the open-source SFST-based SMOR morphology developed at the IMS Stuttgart (Schmid et al. [1]). SFST is a meta-language for finite-state transducers with an efficient compiler that has been specially designed for computational morphology (Schmid [3]).

The SMOR morphology builds on previous work done in DMOR (Schiller [4]) and DeKo (Heid et al. [5]) and provides an elaborate set of rules for German inflection, compounding, and derivation. Its implementation is in the spirit of Karttunen & Beesley ([6]): Morphological alternations are handled by a cascade of (conditional) replacement rules, inflectional paradigms and graduation of adjectives are defined via a set of continuation classes, restrictions on rules are given by means of filter rules, and simplex units are concatenated to form more complex units.

2. Bootstrapping a Morphological Analyzer

As can be expected, the coverage of SMOR with the distributed 1,000-entry lexicon is small. Therefore, we performed two steps to increase it:

- a. Addition of new (mostly) simplex entries and affixes
- b. Addition or modification of morphotactic rules and filters

In fact, (a) and (b) are interdependent. For instance, compositional and derivational stems can be either generated automatically from base stems by specific rules, or they can be defined separately in the lexicon.

Our main goal, thus, is the (semi)automatic construction of a large computational lexicon for SMOR that can be freely distributed and that covers a large part of standard Modern German. The points we want to address are the following:

- 1 How do we obtain the inflectional and derivational information for SMOR from a machine readable dictionary (MRD) automatically?
- 2 How do we manage the lexical data and make it reusable for other researchers or developers of NLP software?
- 3 What effort in a cycle of testing and reengineering is needed to define a finite-state analyzer for the basic vocabulary of standard German?
- 4 How good is the tool in terms of efficiency, accuracy, robustness, and coverage?

2.1. Extracting Morphological Information from an MRD

The Adelung lexicon from 1793 is an early NHD dictionary covering more than 65,000 entries. We used the free edition published by the “Digitale Bibliothek”³ for a

³Adelung - Grammatisch-kritisches Wörterbuch der Hochdeutschen Mundart, available from <http://www.zeno.org/Adelung-1793>.

(semi)automatic extraction of morphological information. Despite some inconsistencies and errors we found, most of the desired entries could be parsed and transformed automatically. We used the dictionary in particular for the extraction of inflectional and derivational information (headword/lemma, gender, inflection paradigm, derivational variants). We defined about 50 rules that map common patterns used by the lexicographer to the corresponding SMOR inflection tag (Schiller et al., [7]). For instance, the pattern *Das Futter, des -s, plur ut nom. sing.* (lining, food) is mapped to the SMOR tag *NNeut_s_0*.

Below you can find an excerpt from Adelung (1793):

1. Das Futter, des -s, plur. ut nom. sing. die Bekleidung eines Körpers von außen oder von innen; [...]
2. Das Futter, des -s, plur. ut nom. sing. 1) Alles, was Menschen und Thieren zur Nahrung dienet; ohne Plural [...]

Figure 1. Excerpt from Adelung (1793) for the German homonym *Futter*(lining, food)

German umlaut variants and diminutive forms are explicitly encoded in Adelung and could be mapped automatically to SMOR inflection classes. In the case of homonyms, (cf. the entry for *Futter* above), two or more entries have been collapsed into one single class. As an initial result, we obtained a lexicon with 32,700 entries for inflecting words fully automatically (see Table 1).

The Adelung lexicon was used as a starting point for our NHD computational lexicon. For a successful mapping we defined some rules to accommodate for the diachronic changes of some lemmas [*th* → *t* (e.g., *thun*), *ey* → *ei* (e.g., *seyn*), *c* → *k* (*cabeljau*)] and normalized the special character encoding (e.g., *Fußtrabánt* → *Fußtrabant*). Some words have changed their inflection pattern in the course of time. These cases, as far as they are documented in the literature, have been corrected manually.

Finally, we compiled our Adelung transducer, comprising SMOR and the new lexicon. We invested only minimal manual effort in building the lexicon, basically for creating further lexicon entries for non-inflecting word classes.

2.2. Managing the Lexical Data and Making It Reusable

As it is difficult for developers to add, modify, remove, and convert lexical data within the original SMOR lexicon format, we defined an exchange format that is independent of the specific finite-state platform. We then implemented a database with an enhanced user interface that is more convenient for lexicographic work than a simple text editor. Thus, we set up a database table in PostgreSQL for nouns, verbs, and adjectives which contain all information necessary for creating an SMOR-based finite-state automaton plus additional lexicographic information (columns for *simplex*, *source*, *frequency*, *editor*). Only recently, we added a fifth lexicographic field stating the "gold standard analysis" for each word. This enables us to produce the correct analysis for ambiguous segmentations which our transducer was not able to handle correctly before.

The database helped us greatly to ensure consistency among entries which otherwise might have caused the transducer to interrupt. We also designed a Relax NG Schema for an XML-based version of our lexicon. The data from the database is converted to XML,

```

<smor> <BaseStem>
  <Lemma>Atlas</Lemma> <Stem>Atlanten</Stem>
  <Pos>NN</Pos> <Origin>nativ</Origin>
  <InfClass>NMasc/Pl</InfClass> <Frequency>676</Frequency>
</BaseStem> </smor>

=> <Base_Stems>Atlas:n<>:t<>:e<>:n<NN><base><nativ><NMasc/Pl>

```

Figure 2. Entry for the plural form of *Atlas* (*atlas*) in XML and SMOR lexicon format

joined with the XML-based data, validated against the Relax NG scheme, and finally transformed into the SMOR lexicon format with an XSLT stylesheet (see Fig. 2).

2.3. Cycle of Testing and Reengineering

For our training set we used the DeReWo list of the 30,000 most frequent German words. We analyzed the DeReWo list with the Adelung transducer and received an analysis for more than 27,000 words. We looked at all 30,000 DeReWo word analyses and tried to improve results (adding missing base stems, correcting false inflection classes or features). Those new entries were verb particles (*ab-*, *vor-*, *zu-*, etc.), derivation or compound stems (*Kriminal-*, *Kirch-*, *Bio-*, etc.), and new base stems. We also added some missing inflection classes [e.g., NMasc_es_er for *Geist-Geistes-Geister* (ghost)]. Some words could not be analyzed because of missing word formation rules [e.g., suffix rules for noun derivations like *StudentIn/Innen* (male and/or female student(s)), or rules for the derivation of compound stems from base stems (*Peterskirche* (Peter's church))].

As most complex words can be derived automatically once all items have been defined in the lexicon - and German is very productive in compounding and derivation - we concentrated on specifying only new lexical entries for simplexes. The overall approach was to create a new entry only when the morphological analysis of the word was wrong. We found that manually creating the simplex units together with their required features and going through 30,000 test analyses was a time-consuming task. In the worst case, an intensive study of the documentation and software code is required, which would not have been possible without knowledge of the finite-state formalism.

Fine-tuning of the morphological tool also involved the handling of certain stems and affixes that were likely to produce ambiguities. To exclude wrong analysis due to segmentation errors (e.g., *Tee-nager* instead of *Teen-ager*), we decided to include these complex words in the lexicon. This is justified because the DeReWo list consists of high-frequency lexicalized terms, where such ambiguities are unlikely. Additionally, we reduced overgeneration by assigning the tag <NoDef> or <Initial> to some morphemes, e.g., mostly short or antiquated words like *Tand* (bauble), or *Ei* (egg). These tags are used in SMOR to impose restrictions on the productivity of certain compound or derivation stems. Finally, we used the transducer to generate tables with the complete listing of an inflectional paradigm. In this way, most errors in our lexicon could be eliminated by the collaborative work of colleagues testing the data.

2.4. Test Results and Statistics

The Morphisto transducer lexicon is minimal in the sense that only those entries have been included that are needed to analyze our training set. The final (minimal) Mor-

phisto transducer lexicon comprises approximately 18,200 entries, which are all manually checked.

Moreover, we included all word formation affixes described in *grammis*⁴ to complete our final lexicon. Compiling a transducer lexicon with approx. 18,200 entries together with the grammar rules takes approximately 2:5h compiled on a dual-core Pentium D 3.4 Ghz with 8 GB RAM running a Linux2.6.16 kernel for x86_64 architecture. The word class statistics for the Adelung and Morphisto lexicons are shown in Table 1.

Lexicon	Adelung	Morphisto
Base Stems	32,152	17,339
- Nouns	20,605	7,833
- Proper Nouns	2	1,053
- Verb Stems	7,426	4,300
- Adjectives	4,061	3,178
- Adverbs	2	781
- Closed Word Classes	28	190
Derivation Stems	63	67
Compound Stems	30	181
Prefix Stems	94	213
Suffix Stems (Derivation Rules)	404	410
Total	32,743	18,210

Table 1. Number of lexical items in the transducer lexicon

The transducer performed best on our training set, the DeReWo lemma list, with a precision of 95% and a coverage of 99%, as most base stems for frequent words are included in the lexicon. We chose the analysis with the smallest number of morphemes⁵, in order to reduce the number of segmentation errors. This strategy, however, blocks further decomposition of complex words stored entirely in the lexicon, which is fine for lexicalized words, but generally weakens recall.

For a second test, we randomly selected subsets of 100 words in different frequency ranges provided by the open-source spell checker ispell (Jacke, 2006). All test words chosen from ispell are unknown to Morphisto and include many complex word formations. For calculating the frequency classes of the word forms in the ispell list, we used the formula devised for the DeReWo lemma list⁶. We manually checked all test analyses for each frequency class and marked the analyses as follows:

- True Positives (TP) = All analyses which match the manual choice (gold standard)
- False Positives (FP) = All other incorrect analyses generated by the tool
- False Negatives (FN) = All correct analysis that are missing in the results

Detailed test results on the precision and recall rates⁷ are given in Table 2. While the Morphisto transducer performs well on high-frequency words, the number of

⁴<http://hypermedia.ids-mannheim.de/pls/public/gramwb.ansicht>

⁵This is a built-in function in SMOR

⁶ $FrequencyClass(Lemma) := \lfloor \log_2 (f(MostFreqLemma, e.g.'der') \div f(Lemma)) \rfloor$

⁷ $Precision = TP \div (TP + FP) * 100\%$; $Recall = TP \div (TP + FN) * 100\%$

spurious ambiguities and wrong or missing analyses increases in the lower frequency ranges. On average, the precision rate of 96.93% is comparable to the results presented in Schmid et al. (2004). As we decided to exclude proper names from the test set, the recall rate of 91.09% is accordingly higher.

All in all, the number of incorrect analyses is low. Many false positives result from ambiguous compound constituents or in combination with linking elements. For example, the homonym *Fest/fest* (party/locked) in the word *Feststellschraube* (locking screw) leads to a false analysis. Likewise, the word *sätzen* in the compound *Zeichensätzen* can be segmented into *Zeichen* (character) followed by *Sätzen* (set) or followed by a linking *s* and the verb *ätzen* (to etch).

False negatives are often due to an incomplete lexicon. About 9% of our test data has not been analyzed because of missing entries in the lexicon. For instance, the German word *allernächster* (the nearest), has been analyzed as a compound *All* (universe) and *Ler* (shade) followed by a superlative of *nah* (near). As the prefix *aller* is not stored in the lexicon, the correct analysis is missing.

Frequency Classes	Precision	Recall	F-Measure
$F_0 - F_4$	100.00	100.00	100.00
$F_5 - F_8$	99.63	99.63	99.63
$F_9 - F_{12}$	98.74	87.71	92.90
$F_{13} - F_{16}$	98.25	93.85	95.39
$F_{17} - F_{20}$	93.21	84.36	88.56
$F_{21} - F_{25}$	91.77	81.00	85.33
Average	96.93	91.09	93.63

Table 2. Test results on ispell for different frequency classes

3. Integration into the TextGrid Workbench

TextGrid offers an Eclipse-based rich-client platform as a development environment for all text-centered scientific disciplines. Collaborative work is supported by the grid, where resources (storage and computing power) can be shared easily among scientists over the net. From this environment, Morphisto is accessible through a small set of document-style web services conforming to the WSDL 1.1 standard. In this way, the user is not required to have the SFST toolkit installed and running on her/his local workstation.

We will briefly sketch what kind of web services will be offered to the community:

Lexical Lookup in a Dictionary. The Adelung transducer (see Sect. 2.1) has been integrated into a service for automatic lexical lookup. Likewise, our NHD transducer is used to establish an automatic reference to the German online lexicon *elexiko*⁸ at the IDS.

⁸http://hypermedia.ids-mannheim.de/pls/elexiko/p4_start.portal

Automatic Annotation of German Texts. The Morphisto transducer for standard German (see Sect. 2.3) is the basis for a number of web services in TextGrid: (1) morphological analyzer, 2) lemmatizer, 3) batch lemmatizer, and, 4) generator of inflected forms. The last service supports the lexicographer in specifying new entries - the corresponding SMOR tag will be computed in the background whenever the head constituent is in the lexicon.

Translation of words from one language stage to another. A common difficulty in understanding historical literature is due to the fact that words undergo an evolutionary change in form and meaning and therefore are often misconceived by the scholars living today. For instance, a scholar would have to know that the word form *seyest* is derived from the lemma *seyn* which is related to the lemma *sein* in NHD. In our approach, this is implemented by mapping all Adelung lemmas to their NHD counterparts, which in most cases are identical.⁹

4. Outlook

The goal was to implement a wide-coverage morphological analyzer for standard German. We think that the finite-state framework is perfectly suited for this task. Also, the open-source code of SMOR & SFST is a good starting point. However, for linguists without special knowledge of finite-state technology, it is not straightforward to extend and modify the underlying grammar. Therefore, we have opted to address this issue by implementing a lexical database and lexicon export scripts.

More work is still needed to create a user-friendly interface in Eclipse that allows users to add, create, and compile additional transducers on a user-selected set of lexemes and rules. Furthermore, we plan to improve our morphological tool by adding structure to the output, as linguists generally are not only interested in the segmentation of a complex word, but also in its internal hierarchic structure. To increase the coverage of Morphisto, we also like to integrate morphological guessers. Results are promising, whenever the head constituent is known, because there are many syncretic forms that make it difficult to derive the correct features (e.g., gender) otherwise. Last but not least, we would gradually like to build up a "Gold Standard" for German morphological analysis. A morphological treebank with hierarchical analyses of the DeReWo list would be a valuable resource for both computational linguists as well as more traditionally oriented linguists.

Acknowledgements

TextGrid is partially funded by the German Federal Ministry of Education and Research (BMBF) under the D-Grid initiative by agreement 07TG01A-H. Responsibility for the contents of this publication rests with its authors.

⁹This is done by replacement rules on the Adelung lexicon transducer, where the NHD lemmas constitute the new UPPER language, and Adelung word forms appear on the LOWER side.

References

- [1] H. Schmid, A. Fitschen, and U. Heid. SMOR: A German computational morphology covering derivation, composition, and inflection. In *Proceedings of the IVth International Conference on Language Resources and Evaluation (LREC 2004)*, Lisbon, Portugal, 2004.
- [2] P. Gietz, A. Aschenbrenner, S. Büdenbender, F. Jannidis, M. Küster, Ch. Ludwig, W. Pempe, T. Vitt, W. Wegstein, and A. Zielinski. TextGrid and eHumanities. In Peter Sloot, editor, *IEEE Conference on e-Science and Grid Computing. Amsterdam 4.-6.12.2006*, pages 133–133, Los Alamitos, CA, 2006. IEEE Computer Society.
- [3] H. Schmid. A programming language for finite state transducers. In *Proceedings of the 5th International Workshop on Finite State Methods in Natural Language Processing (FSMNLP 2005)*, pages 308–309, Helsinki, 2005.
- [4] A. Schiller. Deutsche Flexions- und Kompositionsmorphologie mit PC-KIMMO. In R. Hausser, editor, *Linguistische Verifikation. Dokumentation zur ersten Morpholympics 1994*, pages 37–52. Niemeyer, Tübingen, 1996.
- [5] U. Heid, B. Säuberlich, and A. Fitschen. Using descriptive generalisations in the acquisition of lexical data for word formation. In *Proceedings of the 3rd Conference on Language Resources and Evaluation Volume IV*, pages 86–92, Las Palmas de Gran Canaria, Spain, 2002.
- [6] L. Karttunen and K. Beesley. *Finite State Morphology*. CSLI Studies in Computational Linguistics. CSLI Publication, Stanford, 2003.
- [7] A. Schiller. DMOR: Benutzeranleitung. Interner report, Institut für Maschinelle Sprachverarbeitung, University Stuttgart, 2005.

This page intentionally left blank

Subject Index

ambiguous proper names	62	named entity recognition	50
automata compression	110	natural language generation	175
automaton	146, 175	NooJ	110
calendar information	122	OpenFST	23
classification	14	OpenFst library	27
clustering	14	optimality theory	134
complex patterns	23	Perso-Arabic script	215
compression	146	phonology	207
computational German		Portuguese named entity	
morphology	224	recognition	62
computational linguistics	72	ranking	14
constraints	50	rational powers series	14
coordination and ellipsis	62	reduplication	207
coreference resolution	50	regression	14
Croatian	183	regular expressions	199
DAWG	146	regularization	98
electronic dictionaries	110, 146	research infrastructures	3
event extraction	158	Romanization	215
finite automata	199	search engines	23
finite state model	175	semantic calendar expressions	122
finite state programming language	27	semiring(s)	72, 134
finite-state grammar(s)	39, 62, 158	sequential minimization	110
finite-state machines	23, 50	SFST	224
finite-state methods	98	software architecture	3
finite-state technology	207	statistical machine translation	39
finite-state transducer(s)	3, 122, 183, 215, 224	string matching	23
greedy algorithm	146	tagging	50
grid	224	temporal representation	122
GUI	191	text mining	23
information extraction	50, 62	text processing	175
instruction selection	191	timeline	122
Italian	158	toolkit	191
kernels	14	transducers	199
Kleene	27	transliteration	215
language identification	175	tree adjoining grammar	98
large-scale natural language		tree algorithms	191
processing	39	tree automata	191
learning	14	weighted automata	14, 199
local grammar	62	weighted finite state algebra	134
morphological analysis	3	weighted finite state transducers	39
morphology	183, 207	weighted finite-state automata	72
multi-dimensional trees	98	weighted transducers	14
		XML format	199

This page intentionally left blank

Author Index

Ahrenberg, L.	215	Lesaint, F.	199
Beesley, K.R.	27	Lombardy, S.	199
Bischoff, S.T.	207	Maleki, J.	215
Blackwood, G.	39	Maurel, D.	146
Bouchou, B.	146	Mesfar, S.	110
Brunning, J.	39	Mohri, M.	14
Byrne, W.	39	Niemi, J.	122
Cantone, D.	175	Piskorski, J.	v, 158
Ćavar, D.	183	Quernheim, D.	134
Cleophas, L.	191	Ranchhod, E.	62
Cortes, C.	14	Sakarovitch, J.	199
Cristofaro, S.	175	Seeker, W.	134
de Gispert, A.	39	Silberztein, M.	110
Demaille, A.	199	Simon, C.	224
Didakowski, J.	50	Skut, W.	23
Drotschmann, M.	50	Stojanov, T.	183
Duret-Lutz, A.	199	Tanev, H.	158
Eleutério, S.	62	Terrones, F.	199
Faro, S.	175	Tounsi, L.	146
Giaquinta, E.	175	Watson, B.	v
Hanneforth, T.	72	Yaesoubi, M.	215
Hulden, M.	82, 207	Yli-Jyrä, A.	v, 3
Jazbec, I.-P.	183	Zavarella, V.	158
Kasprzik, A.	98	Zielinski, A.	224
Koskenniemi, K.	3, 122		

This page intentionally left blank

This page intentionally left blank

This page intentionally left blank

This page intentionally left blank

This page intentionally left blank